Optimization of Algorithms Using Extensions of Dynamic Programming

Dissertation by
Hassan AbouEisha

In Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy

King Abdullah University of Science and Technology
Thuwal, Kingdom of Saudi Arabia

January, 2017
The dissertation of Hassan AbouEisha is approved by the examination committee.

Committee Chairperson: Mikhail Moshkov
Committee Members: David Keyes, Xin Gao, Endre Boros
ABSTRACT

Optimization of Algorithms Using Extensions of Dynamic Programming
Hassan AbouEisha

We study and answer questions related to the complexity of various important problems such as: multi-frontal solvers of $hp$-adaptive finite element method, sorting and majority. We advocate the use of dynamic programming as a viable tool to study optimal algorithms for these problems. The main approach used to attack these problems is modeling classes of algorithms that may solve this problem using a discrete model of computation then defining cost functions on this discrete structure that reflect different complexity measures of the represented algorithms. As a last step, dynamic programming algorithms are designed and used to optimize those models (algorithms) and to obtain exact results on the complexity of the studied problems.

The first part of the thesis presents a novel model of computation (element partition tree) that represents a class of algorithms for multi-frontal solvers along with cost functions reflecting various complexity measures such as: time and space. It then introduces dynamic programming algorithms for multi-stage and bi-criteria optimization of element partition trees. In addition, it presents results based on optimal element partition trees for famous benchmark meshes such as: meshes with point and edge singularities. New improved heuristics for those benchmark meshes were obtained based on insights of the optimal results found by our algorithms.

The second part of the thesis starts by introducing a general problem where different problems can be reduced to and show how to use a decision table to model such problem. We describe how decision trees and decision tests for this table correspond
to adaptive and non-adaptive algorithms for the original problem. We present exact bounds on the average time complexity of adaptive algorithms for the eight elements sorting problem. Then bounds on adaptive and non-adaptive algorithms for a variant of the majority problem are introduced. Adaptive algorithms are modeled as decision trees whose depth reflects the worst-case time complexity and average depth indicates the average-case time complexity. Non-adaptive algorithms are represented as decision tests whose size expresses the worst-case time complexity. Finally, we present a dynamic programming algorithm that finds a minimum decision test (minimum reduct) for a given decision table.
ACKNOWLEDGEMENTS

First of all, I would like to thank Allah for providing me with the strength to complete this work.

I would like to thank Prof. Mikhail Moshkov, my advisor, mentor and friend, for his continuous support throughout all my stay in KAUST. He was always providing me with the motivation, inspiration and guidance necessary for fulfilling this work. I cannot have hoped of a better advisor than him.

I would like to extend my appreciation to my committee members: Prof. David Keyes and Dr. Xin Gao for their support and help in addition to Prof. Endre Boros (external examiner and director of RUTCOR) for checking my thesis and spending a weekend morning in my defense. I am indebted to King Abdullah University of Science and Technology for allowing me to fulfill my research goals and interests.

I would like to thank our collaborators: Prof. Maciek Paszynski, Dr. Anna Paszynska and Prof. Victor Calo whose experience and vast knowledge with solvers of the $hp$ adaptive finite element method helped us in creating and evaluating our solutions. I am grateful to Prof. Maciek Paszynski for working closely with me and allowing me to visit his group in AGH university.

I would like to thank Igor Chikalov for his help in working with various problems. His incredible problem solving and coding skills were a main source of inspiration.

I would like to thank my colleagues and friends: Dr. Shahid Hussain, Dr. Fawaz Alsolami, Talha Amin, Mohammed Azad and Michal Mankowski for the continuous discussion and collaboration. I would like to express my gratitude to anybody who contributed to this thesis in any direct or indirect way.

Most importantly, I would like to thank my parents for their love, patience and support that were my main drive to success. My sincere thanks also go to my family and friends for their emotional support during all my study period.
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examination Committee Page</td>
<td>2</td>
</tr>
<tr>
<td>Copyright</td>
<td>3</td>
</tr>
<tr>
<td>Abstract</td>
<td>4</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>6</td>
</tr>
<tr>
<td>List of Figures</td>
<td>10</td>
</tr>
<tr>
<td>List of Tables</td>
<td>11</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>13</td>
</tr>
<tr>
<td>1.1 Contribution</td>
<td>17</td>
</tr>
<tr>
<td>1.2 Outline of Thesis</td>
<td>18</td>
</tr>
<tr>
<td>1.2.1 Part I: Optimizing Algorithms for Multi-Frontal Solvers</td>
<td>19</td>
</tr>
<tr>
<td>1.2.2 Part II: Adaptive and Non-Adaptive Algorithms</td>
<td>20</td>
</tr>
<tr>
<td><strong>I Optimizing Algorithms for Multi-Frontal Solvers</strong></td>
<td>22</td>
</tr>
<tr>
<td><strong>2 Element Partition Trees: Main Notions</strong></td>
<td>24</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>24</td>
</tr>
<tr>
<td>2.2 Meshes and Element Partition Trees</td>
<td>31</td>
</tr>
<tr>
<td>2.2.1 Meshes</td>
<td>31</td>
</tr>
<tr>
<td>2.2.2 Mesh and Line Parameters</td>
<td>34</td>
</tr>
<tr>
<td>2.2.3 Element Partition Trees</td>
<td>35</td>
</tr>
<tr>
<td>2.2.4 Cost Functions for Element Partition Trees</td>
<td>37</td>
</tr>
<tr>
<td><strong>3 Multi-Stage Optimization of Element Partition Trees</strong></td>
<td>45</td>
</tr>
<tr>
<td>3.1 Directed Acyclic Graph $\Delta(M)$</td>
<td>46</td>
</tr>
<tr>
<td>3.2 Cardinality of the Set $Etree(G,M)$</td>
<td>48</td>
</tr>
<tr>
<td>3.3 Optimization Procedure</td>
<td>50</td>
</tr>
</tbody>
</table>
3.4 Multi-Stage Optimization ........................................... 56
3.5 Totally Optimal Trees ............................................. 56
3.6 Experimental Study of Three Types of Meshes .................. 57
  3.6.1 Experimental Settings ........................................ 57
  3.6.2 Optimal Element Partition Trees ............................ 59
  3.6.3 Totally Optimal Element Partition Trees ..................... 62

4 Bi-Criteria Optimization of Element Partition Trees .......... 69
  4.1 Tools ............................................................. 70
  4.2 Bi-Criteria Optimization ......................................... 74
  4.3 Experimental Results ........................................... 80

II Adaptive and Non-Adaptive Algorithms ......................... 86

5 Models of Computations ........................................... 87
  5.1 Introduction ..................................................... 88
  5.2 Problems as Decision Tables ................................... 89
  5.3 Decision Trees .................................................. 91
    5.3.1 Optimization of Decision Trees ............................. 93
  5.4 Decision Tests .................................................. 93
    5.4.1 Optimization of Decision Tests ............................. 94

6 Fundamental Combinatorial Problems ........................... 95
  6.1 Sorting .......................................................... 95
    6.1.1 Introduction ............................................... 96
    6.1.2 Main Results ............................................... 97
    6.1.3 Tools ....................................................... 100
  6.2 Modified Majority Problem ................................... 105
    6.2.1 Introduction ............................................... 105
    6.2.2 Main Notions and Results ................................ 106

7 Optimization of Decision Tests ................................ 110
  7.1 Introduction .................................................... 110
  7.2 Methodology .................................................... 112
    7.2.1 Main Notions ............................................... 112
    7.2.2 Algorithm .................................................. 113
  7.3 Experiments ..................................................... 116
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Example of mesh</td>
<td>32</td>
</tr>
<tr>
<td>2.2</td>
<td>Element partition tree for the mesh in Fig. 2.1</td>
<td>36</td>
</tr>
<tr>
<td>3.1</td>
<td>$P_2$ mesh</td>
<td>59</td>
</tr>
<tr>
<td>3.2</td>
<td>$E_2$ mesh</td>
<td>59</td>
</tr>
<tr>
<td>3.3</td>
<td>$A_2$ mesh</td>
<td>60</td>
</tr>
<tr>
<td>4.1</td>
<td>Pareto front and relationship: $time_{ASAP}^{\infty,4,10^7} vs memory_{ASAP}^4$ for $P_4$</td>
<td>84</td>
</tr>
<tr>
<td>4.2</td>
<td>Pareto front and relationship: $time_{ASAP}^{\infty,4,10^7} vs memory_{ASAP}^4$ for $E_4$</td>
<td>84</td>
</tr>
<tr>
<td>4.3</td>
<td>Pareto front and relationship: $time_{ASAP}^{\infty,4,10^7} vs memory_{ASAP}^4$ for $A_4$</td>
<td>85</td>
</tr>
<tr>
<td>6.1</td>
<td>Trivial decision tree</td>
<td>101</td>
</tr>
<tr>
<td>6.2</td>
<td>Aggregated decision tree</td>
<td>101</td>
</tr>
</tbody>
</table>
LIST OF TABLES

3.1 Number of optimal element partition trees for mesh $P_k$ and cost function $time_{KB}^{1,p,107}$ ................................................................. 61
3.2 Number of optimal element partition trees for mesh $P_k$ and cost function $time_{ASAP}^{1,p,107}$ ................................................................. 62
3.3 Number of optimal element partition trees for mesh $E_k$ and cost function $time_{KB}^{1,p,107}$ ................................................................. 63
3.4 Number of optimal element partition trees for mesh $E_k$ and cost function $time_{ASAP}^{1,p,107}$ ................................................................. 64
3.5 Number of optimal element partition trees for mesh $A_k$ and cost function $time_{KB}^{1,p,107}$ ................................................................. 64
3.6 Number of optimal element partition trees for mesh $A_k$ and cost function $time_{ASAP}^{1,p,107}$ ................................................................. 65
3.7 Existence of totally optimal element partition trees for instances of $P_k$ meshes where $1 \leq k \leq 10$ and $1 \leq p \leq 5$. ........................................... 65
3.8 Existence of totally optimal trees for $P_k$ relative to $time_{KB}^{1,p,107}$, $time_{KB}^{\infty,p,0}$ 66
3.9 Existence of totally optimal element partition trees for instances of $E_k$ meshes where $1 \leq k \leq 10$ and $1 \leq p \leq 5$. ........................................... 66
3.10 Existence of totally optimal trees for $E_k$ relative to $time_{KB}^{1,p,107}$, $time_{KB}^{\infty,p,107}$ 67
3.11 Existence of totally optimal element partition trees for instances of $A_k$ meshes where $1 \leq k \leq 10$ and $1 \leq p \leq 5$. ........................................... 67
3.12 Existence of totally optimal trees for $A_k$ relative to $time_{KB}^{1,p,107}$, $time_{KB}^{\infty,p,0}$ 68
4.1 Size of Pareto front: $time_{ASAP}^{\infty,p,107}$ vs $memory_{ASAP}^{p}$ for $P_k$ ......................... 81
4.2 Size of Pareto front: $time_{ASAP}^{\infty,p,107}$ vs $memory_{ASAP}^{p}$ for $E_k$ ......................... 81
4.3 Size of Pareto front: $time_{ASAP}^{\infty,p,107}$ vs $memory_{ASAP}^{p}$ for $A_k$ ......................... 82
4.4 Size of Pareto front: $time_{KB}^{\infty,p,107}$ vs $memory_{KB}^{p}$ for $P_k$ ................................. 82
4.5 Size of Pareto front: $time_{KB}^{\infty,p,107}$ vs $memory_{KB}^{p}$ for $E_k$ ................................. 83
4.6 Size of Pareto front: $time_{KB}^{\infty,p,107}$ vs $memory_{KB}^{p}$ for $A_k$ ................................. 83
4.7 Execution time of bi-criteria optimization for $A_k$ mesh with respect to $time_{ASAP}^{\infty,p,107}$ and $memory_{ASAP}^{p}$ ........................................... 85
6.1 Results for sorting $n = 2, 3, 4, 5$ elements. 

6.2 Results for sorting $n = 6, 7, 8$ elements.

7.1 caption
Chapter 1

Introduction

Dynamic Programming is a general algorithmic tool that can be used with a variety of problems [1]. The term dynamic programming was introduced by Richard Bellman during his work at RAND in 1952 [2–4]. A given problem may be solved using dynamic programming if it exhibits the optimal substructure property (principle of optimality) and it can be decomposed into overlapping smaller subproblems. Dynamic programming is applied extensively in many fields such as: bioinformatics, economics, transportation management and decision making. Different applications of dynamic programming are illustrated in [5].

A dynamic programming algorithm starts its work by decomposing the original problem into smaller subproblems, then solves those subproblems and combines their solutions into a solution of the original problem. The dynamic programming method solves each subproblem only once and stores its solution for later retrievals. The dynamic programming approach is usually used with optimization problems. In such cases, the problem has many solutions with different values of the considered criterion and we are looking for an optimal solution [6].

One of the main questions considered when studying a given problem is the complexity of the problem under study. The question is concerned with identifying the best algorithm that solves this problem with respect to some criterion. The most famous criteria are time and memory complexity. There are two approaches for studying the complexity of a given problem. One starts by designing an algorithm to solve the problem and then this algorithm complexity becomes an upper bound on
the problem complexity. We work then to improve the existing algorithm or create a new one with better complexity according to the study criterion. This approach is what is well known as algorithm design and analysis.

Another approach for answering this question is to reason about the problem itself rather than the algorithm. In other words, we consider the hardness of the problem. We are willing to say that any algorithm that solves this problem will have at least a specific cost. In other words, we are enforcing some lower bound on all algorithms that solve this problem. This area of study is a branch of complexity analysis.

We consider applications of the dynamic programming technique in the area of complexity theory. We are interested in automatically optimizing algorithms for different problems. In other words, we seek an optimal algorithm for a given problem in the space of all algorithms that correctly solves this problem. The key idea of considering all correct algorithms for a given problem is to define a model of computation that represents an algorithm for this problem. The next step is to define the aspect (criterion) for which we consider an algorithm to be optimal. We define cost functions that reflect the optimization criterion of the studied algorithm. This may include different measures such as time complexity, space complexity, energy efficiency and communication complexity.

It is common when considering the complexity of a given problem to specify its size. For example, we may study the best average complexity of finding the third smallest number among a set of 10 numbers. In most cases, the tools used run in exponential time in the worst case so we cannot apply them directly for large instances of the studied problems. An exception to this are the algorithms we introduce for finding optimal algorithms for multi-frontal solvers that run in polynomial time in the problem size.

The motivation of designing and applying these tools to optimize algorithms for different instances of problems is extensive. They may be used as a tool to answer
long standing open theoretical questions as is the case with the average depth of a
comparison-based algorithm to sort a sequence of eight elements. We discuss this
problem in Section 6.1. Another possible application is studying optimal algorithms
with the aim of designing a heuristic algorithm that learns from the behavior of the
optimal algorithm. Finally, we may find interesting results for small instance of the
problems that can give insights into generalizing those results to all instances of the
problem. We have this situation with a variant of the majority problem discussed in
Section 6.2. We applied the tools for understanding what is the exact complexity for
this problem when the size of the input ranges from one to ten. We then succeeded
in generalizing the results for an input with arbitrary size \( n \).

We apply this approach to a variety of interesting problems. The first one is an
application in the field of finite element method in which we search for an optimal
algorithm to find a solution for a partial differential equation (PDE) approximated
by a given mesh. We explore various classes of meshes with different degrees of
complexity. We model algorithms for solving the PDE approximated by a finite
element mesh with element partition trees. We design algorithms for multi-stage
optimization of element partition trees. These algorithms allow us to represent the set
of all element partition trees for a given mesh compactly, count the cardinality of such
set without enumerating its members and finally optimize the set of element partition
trees with respect to a given cost function. The result of the procedure of optimization
is a subset of optimal element partition trees rather than a single one. For strictly
increasing cost functions (see Section 2.2.4), our optimization algorithm finds the
whole set of optimal solutions. This calls for a multi-stage optimization process
where different cost functions may be used in each stage. Different improvements for
current solvers for a wide class of finite element meshes were obtained based on our
results [7, 8].

Multi-stage optimization can be very useful when we have element partition trees
that are optimal with respect to different criteria simultaneously. Unfortunately, this is not the case in many situations. Usually, there are trade-offs between different cost functions like: time and memory or power consumption and time. For power consumption, we estimate the solver’s cost as numbers of floating point operations and memory transfer operations. Given two cost functions, we consider the set of points corresponding to all possible element partition trees for a given mesh in the following way. For a point representing a given element partition tree, the first coordinate of this point represents the cost of this tree with respect to the first cost function and the second coordinate denotes the cost of this tree with respect to the second cost function. We define a Pareto optimal (non-dominated) point of this set as a point where there is no other point whose coordinates are at most coordinates of current point. We propose a dynamic programming based method to find the set of Pareto optimal points with respect to a given mesh and two cost functions. In addition, this method enables us to create an element partition tree corresponding to each Pareto optimal point. We refer to those element partition trees as Pareto optimal trees. Such technique allows us to study relationship between different cost functions and study Pareto optimal algorithms (element partition trees) that can help creating balanced heuristics. An improved iterative solver is proposed based on our results in [9].

In addition, we use this approach with a variety of fundamental problems in algorithms such as: sorting and majority. These problems are well studied in the complexity literature but we intend to fill some of the gaps in complexity bounds. These algorithmic problems can be modeled using decision tree as our model of computation. We prove that the minimum average depth of a decision tree for solving the eight element sorting problem is $\frac{620160}{8!}$ [10] which is an open problem discussed in [11]. We show that the set of decision trees for eight elements sorting with minimum average depth is a subset of the set of optimal decision trees for this problem with minimum depth. We obtain these results with the help of DAGGER [12] as it
provides functionality of multi-stage optimization of decision trees. Finally, we consider a modified majority problem where we seek the first majority element (if exists) in a given binary sequence of \( n \) elements. We prove that the minimum depth of a decision tree solving this problem is \( n - 1 \) and the minimum cardinality of a decision test modeling a non-adaptive algorithm for this problem is \( n - 1 \). We formulate our hypotheses using experiments involving multi-stage optimization of decision trees as proposed in [13,14] and implemented in DAGGER [12].

We consider tests and reducts as a third different model of computation. This model of computation can be used to express non-adaptive (oblivious) algorithms. That kind of algorithms does not change its behavior according to intermediate results of its steps. We proposed two algorithms for finding a minimum reduct [15,16], however in this thesis we discuss only one of them [16]. These algorithms are important as reduct is a fundamental concept in rough set theory [17–19]. Furthermore, it has many applications in knowledge representation, feature selection and data mining. The proposed algorithm uses multi-stage optimization of decision trees as a final step of its work and we used the DAGGER software system [12] for executing this task during the computer experiments.

1.1 Contribution

The main contribution for this thesis can be divided into three categories:

- Theoretical:
  - Algorithms for multi-stage optimization of element partition trees.
  - Algorithms for bi-criteria optimization of element partition trees.
  - Exact algorithm for finding minimum decision tests (reducts).
  - Different results about bounds for combinatorial problems like: sorting of eight elements and a modified majority problem.
• Software development:
  
  – A tool for multi-stage optimization of element partition trees based on our proposed algorithm.
  
  – A tool for bi-criteria optimization of element partition trees.
  
  – A tool for finding minimum tests.

• Experimental:

  – A library of optimal element partition trees that were studied and several improvements on current solvers based on them were designed by our computational sciences collaborators.
  
  – A library of Pareto optimal element partition trees.
  
  – Experiments on finding minimum tests for 23 datasets from UCI Machine Learning Repository [20].

1.2 Outline of Thesis

The thesis is divided into two main parts followed by a concluding chapter. Both parts demonstrate the proposed approach of this thesis. This approach starts by defining a model of computation that describes a given class of algorithms for a given problem. It then captures the complexity criteria of interest with the use of cost functions. Finally, dynamic programming algorithms are used to find an optimal algorithm (represented by a model of computation) with respect to the cost function of interest.

We divide the thesis into two parts as the first part proposes a non-traditional model of computation that we use to model a given class of algorithms for multifrontal solvers along with new optimization tools. On the other hand, the second part uses traditional models of computation such as: decision trees and decision tests
however it presents new results about complexities of traditional problems in addition to a new dynamic programming based optimization tool for reducts.

1.2.1 Part I: Optimizing Algorithms for Multi-Frontal Solvers

This part of the thesis studies multi-frontal solvers for the \( hp \)-adaptive finite element method. It provides a mathematical model of the problem and a new model of computation (element partition tree) to describe classes of algorithms for multi-frontal solvers. We discuss different methods that we create based on dynamic programming to find optimal algorithms for multi-frontal solvers. It consists of three chapters featuring algorithms and results related to the problem of element partition tree optimization.

Chapter 2: Element Partition Trees: Main Notions

This chapter introduces a brief background about the \( hp \)-adaptive finite element method along with the motivation. It then presents element partition trees as a model of computation to describe a class of algorithms for multi-frontal solvers. Furthermore, it highlights the basic notions used and shows the theoretical framework of our optimization algorithm such as cost functions.

Chapter 3: Multi-Stage Optimization of Element Partition Trees

This chapter describes algorithms for multi-stage optimization of element partition trees based on dynamic programming. We present results about optimal element partition trees for various classes of finite element meshes. Currently, we consider only three classes of meshes that we may vary their degree of complexity in addition to three cost functions.
Chapter 4: Bi-Criteria Optimization of Element Partition Trees

Algorithms for investigating the relationships between different pairs of cost functions are established in this chapter. Those algorithms are designed to find the set of Pareto optimal points and can construct a Pareto optimal element partition tree corresponding to each Pareto optimal point. Moreover, we suggest how to utilize the set of Pareto optimal points to depict relationships between pairs of cost functions.

1.2.2 Part II: Adaptive and Non-Adaptive Algorithms

The second part discusses tools and results for adaptive algorithms and non-adaptive (oblivious) algorithms. Adaptive algorithms are modeled using decision trees while non-adaptive algorithms are represented by decision tests.

Chapter 5: Models of Computation

This chapter shows a general problem that can be modeled using a decision table. It defines decision trees and decision tests for decision table. We show that decision trees for a decision table representation of the general problem introduced model naturally correct adaptive algorithms for the problem. Similarly, we argue that decision tests for the same table represent non-adaptive algorithms for the corresponding problem.

Chapter 6: Fundamental Combinatorial Problems

This chapter describes results for fundamental problems in computer science such as: sorting, and a modified majority problem that both can be modeled using decision tables. We show how different algorithms for each problem can be modeled using decision trees. In addition, we describe results about the problem that were well known and present our new results.
Chapter 7: Optimization of Tests

This chapter presents an algorithm for finding minimum exact tests for decision tables. We show that this algorithm can be used with medium sized datasets by performing experiments on different UCI Machine Learning Repository [20] datasets.

Chapter 8: Conclusion

This chapter concludes the thesis with a summary of the proposed method, contribution and a list of future work.
Part I

Optimizing Algorithms for
Multi-Frontal Solvers
The first part of this thesis is devoted to the optimization of element partition
trees controlling the LU factorization of systems of linear equations resulting from the
finite element method discretization over two dimensional meshes with rectangular
elements.

We begin by presenting an introduction to the considered problem followed by
the main notions and definitions for the main mathematical concepts that we study
in Chapter 2. They span meshes, element partition trees and cost functions for
element partition trees. We create and study polynomial algorithms for optimization
of element partition trees which use only straight lines for partitioning of rectangular
meshes.

The considered algorithms are based on extensions of dynamic programming and
allow multi-stage optimization of element partition trees relative to different criteria
such as time and memory complexity. We describe the multi-stage optimization
process and the experimental results in Chapter 3.

Finally, we present polynomial time algorithms for bi-criteria optimization of ele-
ment partition trees along with possible applications in Chapter 4.
Chapter 2

Element Partition Trees: Main Notions

The $hp$-adaptive finite element method is a popular approach used to find approximate solutions for partial differential equations (PDEs). Initially, the problem domain is approximated using a mesh of elements. Then the PDE is transformed into a weak form and discretized using basis functions over the vertices, edges and interior of elements. As a result of the discretization process, we obtain a global system of linear equations. Such system can be decomposed into frontal matrices associated with the mesh elements. Solution of the PDE can be obtained by working with those frontal matrices. An element partition tree prescribes the order that the solver considers while working with the frontal matrices. Using this order we obtain an ordering of the rows of the global matrix representing the system of linear equations.

This chapter starts by formally defining the class of meshes under study. We describe the notion of an element partition tree and present an abstract way of defining optimization criteria of element partition trees in terms of cost functions. A definition of a cost function is provided in addition to a few examples of cost functions under study along with some of their properties.

2.1 Introduction

A rectangular mesh can be considered as a finite set of vertical and horizontal line segments, vertices (intersections of line segments) and rectangles bounded by these line segments. An element partition tree is a binary rooted tree that describes the
partition of the mesh by line segments. Cost functions for the element partition
trees are defined based mainly on the topological information of the mesh. We study
the problems of multi-stage and bi-criteria optimization of element partition trees
for rectangular meshes. The considered problems lie at the intersection of discrete
geometry and combinatorial optimization. We begin with a brief discussion of the
problem motivation and assumptions which simplify the considered model.

The finite element method is a widely used approach to find approximate solutions
of partial differential equations (PDEs) specified along with boundary conditions and
a solution domain. A mesh with triangular or rectangular elements is created to
cover the domain and to approximate the solution over it. Then the weak form of
the PDE is discretized using polynomial basis functions spread over the mesh (1 per
vertex, $p$ per edge for some nonnegative integer $p$, and $p^2$ per element interior) for
finite elements of uniform polynomial order $p$. Let $\nu$ be the number of basis functions
used. The approximate solution of the discretized PDE can be represented as a linear
combination of these $\nu$ functions. The coefficients of the linear form can be found as
solutions of a system of linear equations $Ax = b$ constructed based on PDE, boundary
conditions, and the mesh, where $A$ is a square $\nu \times \nu$ matrix.

Any $\nu \times \nu$ matrix $A$ can be represented in the form $PA = LU$ where $P$ is a
permutation matrix which reorders the rows of $A$, $L$ is a lower triangular matrix in
which all elements above the diagonal are zero, and $U$ is an upper triangular matrix
in which all elements below the diagonal are zero [21]. The construction process
of matrices $P$, $L$ and $U$ is called LU factorization (with partial pivoting). We can
rewrite the equation system $Ax = b$ equivalently as $LUx = Pb$. The solutions of this
system can be found in the following way: we solve the equation system $Ly = Pb$ for
$y$ (forward substitution) and, after that, we solve the equation system $Ux = y$ for $x$
(backward substitution).

The number of arithmetic floating point operations (additions, subtractions, mul-
tiplications, and divisions) for the LU factorization in the worst case is $O(\nu^3)$ if we consider standard algorithms (for example, Doolittle algorithm [22]) and $O(\nu^{2.3728639})$ if we consider the best known (from theoretical point of view) modification of Bunch and Hopcroft algorithm [23] based on Le Gall algorithm for matrix multiplication [24].

This is essentially greater than $O(\nu^2)$ for the forward and backward substitution. The above-mentioned Doolittle algorithm is a simple modification of Gaussian elimination. Let us assume for simplicity that it is applicable to $A$ without the reordering of rows. Then the algorithm constructs the matrix $U$ by a sequential elimination of its rows. Let the first $i-1$, $1 \leq i < \nu$, rows are already eliminated: in the first $i-1$ columns, all elements below the diagonal are zero. To eliminate row $i$ we need $(2\nu - 2i + 1)(\nu - i)$ arithmetic floating point operations. To eliminate the first $\kappa$ rows we need $W(\nu, \kappa) = \sum_{i=1}^{\kappa} (2\nu - 2i + 1)(\nu - i) = 2\nu^2\kappa - 2\nu\kappa^2 - \nu\kappa + \frac{2}{3}\kappa^3 + \frac{1}{2}\kappa^2 - \frac{1}{6}\kappa$ floating point operations, and to construct $U$ we need $W(\nu, \nu)$ operations. All elements of $L$ are extracted from the intermediate results of the algorithm work without additional arithmetic floating point operations. All the entries on the diagonal of $L$ are one. Therefore to keep $U$ and $L$ it is enough the same $\nu^2$ memory as for the matrix $A$.

The multi-frontal solver is the state-of-the-art algorithm for solving sparse linear systems resulting from finite element method discretization [25, 26]. This algorithm works in both single- and multi-processor environment, either in the shared-memory [27–29] or distributed-memory [30–32] parallel machines.

The matrix $A$ is constructed from element frontal matrices corresponding to particular finite elements of the computational mesh. The order of factorization is given by an element partition tree in which each node corresponds to a submesh of the initial mesh. In particular, the root corresponds to the whole mesh and the terminal nodes (the leaves) correspond to finite elements. In each internal node, the submesh corresponding to this node is divided into two submeshes which correspond to children of the considered node.
In each terminal node, we perform partial factorization of the matrix for a finite element corresponding to this node. Partially factorized matrices for children of a node (in fact, parts of these matrices known as Schur complements \([33, 34]\)) are combined into a matrix which is partially factorized in this node. These operations are repeated recursively until we reach the root of the element partition tree. Partially factorized matrices must be kept for the following forward and backward substitution. These partially factorized matrices can be combined to yield the LU factorization of the original matrix \(A\).

Partial factorization means the following. We have a \(\mu \times \mu\) matrix \(B\) such that, for the first \(\rho\) rows of \(B\), \(1 \leq \rho \leq \mu\), variables corresponding to these rows can be expressed (in the considered model) in terms of the variables corresponding to the remaining \(\mu - \rho\) rows. These rows are \textit{fully assembled}. In particular, in the matrix corresponding to a finite element at least all variables corresponding to the interior are fully assembled. That is, all interactions of the corresponding basis functions have been fully captured at the present level. Thus, they can be expressed in terms of variables corresponding to edges and vertices of the finite element. The algorithm eliminates the first \(\rho\) rows of \(B\) as the Doolittle algorithm does, keeps results of partial factorization in memory, and sends to the parent node the submatrix obtained at the intersection of the last \(\mu - \rho\) rows and the last \(\mu - \rho\) columns (Schur complement).

To simplify the model under consideration, we neglect the complexity of forward and backward substitution, and study only the complexity of the LU factorization. We also neglect the complexity of combining the Schur complements into a matrix which is essentially less than the complexity of partial factorization of this matrix. We assume that the Doolittle algorithm is applicable to the partial factorization of each one of the considered matrices without reordering of rows. We use \(W(\mu, \rho)\) as the number of arithmetic floating point operations for partial factorization of a \(\mu \times \mu\) matrix with \(\rho\) fully assembled rows. We assume that to keep results of partial
factorization of a $\mu \times \mu$ matrix we need $\mu^2$ memory, and that we should send $(\mu - \rho)^2$ numbers (corresponding to the Schur complement) to the parent node.

The computational complexity of a multi-frontal solver implementation depends on the element partition tree used. The aim of this work is to optimize element partition trees relative to different cost functions that characterize time or memory complexity. In the general case, when a given mesh can be partitioned in an arbitrary way, the problem of partitioning this mesh to minimize the fill-in is NP-hard [35]. In our case, we only use partitions by straight line segments. We prove that the considered optimization problems have polynomial time complexity.

In this thesis, we consider a set of rectangular meshes which allow us to describe and study known “benchmark” meshes representing various point and edge singularities. We study a class of element partition trees obtained by recursive partitioning of the mesh along straight line segments. We define two elimination strategies. In the first one, at each elimination step all fully assembled degrees of freedom are eliminated. We call this strategy \textit{as soon as possible} and denote it ASAP. The second strategy eliminates all degrees of freedom from a mesh (submesh) with respect to the boundary. This strategy is called \textit{keep boundary} and we denote it KB. KB assumes that the considered mesh is a submesh of another global rectangular mesh that we seek a good element partition tree for. This submesh usually describes a part of the domain with specific kinds of singularities. The major difference between both cost functions is that the KB cost function assumes that rows associated with the domain boundary are kept, that is they are not eliminated until the root of the elimination tree, so they can be interfaced with the remaining parts of the global mesh. The ASAP cost function assumes that the rows associated with the boundary can be eliminated as soon as possible, since the processed mesh is considered as the global one.

We consider three different cost functions associated with each type of elimination
strategy. In particular, we model the memory requirements for each elimination strategy. We also analyze the computational complexity (a proxy of it is execution time) for single- and multi-processor elimination. We introduce the notion of a strictly optimal element partition tree. This is a tree which is optimal for the input mesh and, for each node of the tree, the subtree with root in this node is also optimal for the corresponding submesh. Except for the two cost functions that characterize time for multi-processor computations, the set of strictly optimal elimination trees for the rest of the cost functions coincides with the set of optimal element partition trees. For the two cost functions characterizing the time of multi-processor computations, the set of strictly optimal element partition trees is a subset of the set of optimal element partition trees.

To optimize element partition trees, we use some extensions of dynamic programming which allow us to describe the set of strictly optimal element partition trees by a directed acyclic graph whose nodes are submeshes of the initial mesh. Dynamic programming also allows us to count the number of strictly optimal trees, and to understand either the obtained trees or their prefixes if the strictly optimal trees are too large. We also have the possibility to perform multi-stage optimization of element partition trees relative to a sequence of cost functions.

We may be interested in finding a good element partition tree with respect to two criteria. For example, an element partition tree that results in small number of floating point operations by the solver and does not consume much intermediate memory is interesting. The considered criteria are sometimes conflicting so we aim to find element partition trees with acceptable trade-off between the studied cost functions.

We present a bi-criteria optimization algorithm for element partition trees. Such algorithm allows us to construct the set of Pareto optimal points corresponding to values of the two studied cost functions on element partition trees for a given mesh.
We refer to this set of points as the Pareto front.

The Pareto front is used to depict the relationship between the two studied criteria. If the Pareto front consists of only one point then totally optimal element partition trees for both cost functions exist. In other words, we have at least one element partition tree that has optimal values of both considered criteria simultaneously.

All the considered algorithms have polynomial time complexity in terms of the input mesh size. We implement these algorithms in Java.

We use the obtained software to study three families of meshes describing different kinds of singularities: point singularities, edge singularities, and point-edge singularities. For these meshes, we investigate the number of optimal element partition trees and the existence of totally optimal element partition trees which are optimal relative to a number of cost functions simultaneously. We use the bi-criteria optimization algorithm to compute the Pareto fronts for these meshes.

The KB mode of the optimization algorithms can be used in large meshes. KB improves the performance of solvers by constructing optimal element partition trees for relatively small but essentially irregular submeshes of the global mesh.

The ASAP optimization algorithm is a research tool which helps us understand the behavior of optimal element partition trees that use only straight line segments as separators. The comparison of the optimal element partition trees with the trees constructed by the known heuristics (MUMPS [36] solver with METIS [37] library) allowed us to propose a new heuristic [7,8] which outperforms existing state-of-the-art alternatives.

The first attempt to analyze mathematically the considered dynamic programming algorithms was done in [38]. The investigated model was less accurate, we did not distinguish ASAP and KB modes. We did not consider the notion of strictly optimal element partition trees, and worked only with a cost function which characterizes the time of single-processor computation.
This chapter discusses the main notions connected with meshes, element partition
trees, and cost functions.

2.2 Meshes and Element Partition Trees

We describe the class of finite element meshes studied, and define the notions of an
element partition tree and a cost function for element partition trees.

2.2.1 Meshes

The class of meshes investigated is constructed as follows. We start with a rectangle.
We call the sides of this initial rectangle boundary sides and its vertices are called
corners. Further vertical and horizontal straight line segments (we refer to these
segments including sides of the initial rectangle as lines) may be added as follows.
We select two points that lie on different existing parallel lines and which can be
connected by a vertical or horizontal line and connect them by adding a line. This
process may be repeated until the desired mesh structure is obtained. After this
construction process is finished, we have a mesh that is mainly a rectangle with a set
of vertical and horizontal lines that lies inside it.

The following example illustrates the description of an instance of this class of
meshes presented in Fig. 2.1. Initially, the rectangle boundary $ABCD$ is specified.
The order of points is assumed to follow counter-clockwise order, and the selection
of the starting point is arbitrary. First, line $EF$ is added and its end points belong
to the horizontal sides of the mesh’s boundary ($AD$ and $BC$). Next, $GM$ is drawn
between $AB$ and $EF$ while $MH$ is drawn between $EF$ and $CD$. Finally, lines $KL$
and $JI$ are added.

As a result of this construction process, we obtain a set of points that are either
vertices of the initial rectangle, endpoints of the lines added or points that result
from the intersection of the lines added. We call this set of points mesh points. Any
segment of a straight horizontal or vertical line in the mesh that connects two different mesh points is called a *mesh line*. A *boundary line* is a mesh line which belongs to a boundary side. A *maximal* mesh line is a mesh line which does not belong to any other mesh line. For example, $KE$, $FK$, $CH$ and $GH$ are mesh lines, $CH$ is a boundary line, and $AB$, $GH$ and $JI$ are maximal mesh lines (see Fig. 2.1).

We define *dividing lines* that are used to partition a given mesh. We denote the set of vertical lines that extend between the borders of a mesh $M$ by $DL_V(M)$, horizontal lines that extend between the borders of $M$ by $DL_H(M)$ and the union of both sets by $DL(M)$. We do not consider vertical border sides of the mesh $M$ among $DL_V(M)$ and similarly for its horizontal border sides. The mesh $M$ can be partitioned using a dividing line $l$ that belongs to the set $DL(M)$. This partitioning step results in two submeshes: $M(l,0)$ that represents the submesh which lies below the horizontal (left of the vertical) line $l$ and $M(l,1)$ denotes the submesh which is above the horizontal (right of the vertical) line $l$. In Fig. 2.1, $DL_V(ABCD) = \{FE\}$, $DL_H(ABCD) = \{GH\}$ and $DL(ABCD) = \{GH, FE\}$. The mesh $ABCD$ can be partitioned using the dividing line $FE$ resulting in the two submeshes: $ABFE = ABCD(FE,0)$ and $EFCD = ABCD(FE,1)$.

We describe an arbitrary *submesh* $N$ of $M$ by a sequence of partitioning steps.
Formally, a submesh $N$ of a mesh $M$ is an expression of the kind

$$N = M(l_1, \delta_1)...(l_n, \delta_n)$$

where $\delta_1, \ldots, \delta_n \in \{0, 1\}$, $l_1 \in DL(M)$ and $l_i \in DL(M(l_1, \delta_1)...(l_{i-1}, \delta_{i-1}))$ for $i = 2, \ldots, n$. Note that $M$ is a submesh by itself as it can be obtained by an empty sequence of partitioning steps.

The resulting submesh is described as follows. First, a dividing line $l_1$ is used to partition the mesh $M$ then the submesh $M(l_1, \delta_1)$ is acquired. The line $l_2$ is a dividing line of this submesh ($l_2 \in DL(M(l_1, \delta_1)))$ which is used to partition $M(l_1, \delta_1)$ again until the desired submesh is obtained, etc. For example, $MHDE = ABCD(EF,1)(MH,1)$. We denote by SUB($M$) the set of submeshes of the mesh $M$ including the mesh $M$. In our example,

$$\text{SUB}(ABCD) = \{BFMG, BIJG, BCHG, ABFE, ABCD, AGME, AGHD, FINK, FIJM, FCLK, FCHM, FCDE, ICLN, ICHJ, KNJM, KLHM, KLDE, NLHJ, MHDE\}.$$ 

The submesh $N$ can be considered as a mesh that is represented as a rectangle, whose sides are called border sides, that may contain a set of vertical and horizontal lines inside it. For the submesh $N$, we can define the notion of dividing lines in a similar way as for $M$, so we can use the notation $DL_V(N)$, $DL_H(N)$ and $DL(N)$. As for the initial mesh $M$, it is possible that a dividing line is constructed indirectly in a series of steps. For example, $DL_V(KLHM) = \{NJ\}$ and $DL_H(FIJM) = \{KN\}$ (see Fig. 2.1).

A submesh $N$ of the mesh $M$ is a unitary submesh if and only if it does not have any dividing lines, i.e., $DL(N) = \emptyset$. We assume that each unitary submesh $N$ of $M$ has a unique identifier $\varphi(N)$. Unitary submeshes correspond to finite elements.
2.2.2 Mesh and Line Parameters

Let $M$ be a mesh. For each mesh line $l$ of $M$, we describe a set $P(M,l)$ of distinct points of the line $l$. If $l$ is a boundary line, then the set of points includes the endpoints of $l$ and any point of $l$ that results from any other mesh line touching $l$. If $l$ is not a boundary line, then the set of points includes all the points of $l$ that result from another mesh line cutting $l$ in addition to the endpoints of $l$. End points of other lines that start or finish on $l$ are not included. That is, a touching line does not generate a point for $l$. For example, $P(ABCD, AD) = \{A, E, D\}$, $P(ABCD, EF) = \{F, M, E\}$. Each pair of consecutive points in $P(M,l)$ forms an edge. The number of edges on a line $l$ is denoted by $E(M,l)$ and $E(M,l) = |P(M,l)| - 1$. For example, $E(ABCD, EF) = 2$ for the mesh in Fig. 2.1.

Let $N$ be a submesh of $M$ and $l \in DL(N)$. The line $l$ represents a common border side of two submeshes: $N(l, 0)$ and $N(l, 1)$. We define now a number of parameters of the submesh $N$ and dividing line $l$ for $N$:

- $EP(M,N,l)$ represents the number of endpoints of $l$ that lie in a boundary side.
- $B(M,N)$ is the number of edges on the border sides of $N$. These edges result from mesh lines that cut the border sides of $N$ or touch them if they are on boundary sides, i.e., border sides of the initial mesh $M$.
- $BE(M,N)$ is the number of edges that lie in the border sides of $N$ which are on the boundary sides, where $0 \leq BE(M,N) \leq B(M,N)$.
- $BV(M,N)$ represents the number of vertices of $N$ that are endpoints of boundary sides, i.e., corners, where $0 \leq BV(M,N) \leq 4$. 
For example in Fig. 2.1, some values of the parameters above are:

\[ EP(ABCD,FIJM,KN) = 0, \quad B(ABCD,FIJM) = 5, \]
\[ EP(ABCD,FCHM,KL) = 1, \quad BE(ABCD,BFMG) = 2, \]
\[ EP(ABCD,ABCD,EF) = 2, \quad BV(ABCD,BFMG) = 1. \]

We denote by \( s(M) \) the number of maximal mesh lines in \( M \). The number \( s(M) \) is the size of the mesh \( M \). For example, \( s(ABCD) = 8 \) for the mesh in Fig. 2.1.

**Lemma 1.** For any mesh \( M \), \(|\text{SUB}(M)| \leq s(M)^4\).

**Proof.** Each submesh of \( M \) is defined by its four border sides, i.e., by four straight lines. The number of possible straight lines is at most the size \( s(M) \) of the mesh \( M \). Therefore \(|\text{SUB}(M)| \leq s(M)^4\).

\[ \square \]

### 2.2.3 Element Partition Trees

An element partition tree is a labeled finite directed tree with a root. We define the notion of an element partition tree for a submesh \( N \) of the mesh \( M \) by induction. Let \( N \) be a unitary mesh. Then there exists only one element partition tree for \( N \) that contains exactly one node which is labeled with \( \varphi(N) \). We denote this element partition tree by \( \text{etree}(\varphi(N)) \). Let \( N \) be a nonunitary mesh. Then any element partition tree for \( N \) can be represented in the form \( \text{etree}(l, \Gamma_0, \Gamma_1) \) where \( l \in DL(N) \), \( \Gamma_\delta \) is an element partition tree for the submesh \( N(l, \delta) \), \( \delta \in \{0, 1\} \), and \( \text{etree}(l, \Gamma_0, \Gamma_1) \) is a tree in which the root is labeled with \( l \), and two edges start from the root which are labeled with 0 and 1 and enter the roots of element partition trees \( \Gamma_0 \) and \( \Gamma_1 \), respectively. We denote the set of all element partition trees for the submesh \( N \) of the mesh \( M \) by \( ET(M,N) \).

Let \( \Gamma \) be an element partition tree for the submesh \( N \) of the mesh \( M \). Any
terminal node (leaf) of this tree is labeled as a unitary submesh of $N$. Any internal node is labeled with a line. Each internal node has exactly two edges that start from it and are labeled with 0 and 1, respectively. Fig. 2.2 shows an element partition tree for the mesh presented in Fig. 2.1.

We now associate to each node $v$ of the element partition tree $\Gamma$ a submesh $N_{\Gamma}(v)$ of the submesh $N$. If $v$ is the root of $\Gamma$ then $N_{\Gamma}(v) = N$. If $v$ is not the root and the path from the root to $v$ consists of nodes labeled with lines $l_1,\ldots,l_m$ and edges labeled with the numbers $\delta_1,\ldots,\delta_m$, respectively, then $N_{\Gamma}(v) = N(l_1,\delta_1)\ldots(l_m,\delta_m)$.

For each internal node $v$ of $\Gamma$, this node is labeled with a line from $DL(N_{\Gamma}(v))$. For each terminal node $v$, the submesh $N_{\Gamma}(v)$ is a unitary mesh and $v$ is labeled with the identifier $\phi(N_{\Gamma}(v))$ of $N_{\Gamma}(v)$.

For a node $v$ of $\Gamma$, we denote by $\Gamma(v)$ the subtree of $\Gamma$ with the root in $v$. Thus, $\Gamma(v)$ is an element partition tree for the submesh $N_{\Gamma}(v)$.

Let $M$ be a mesh and $N$ be a nonunitary submesh of $M$. For each $l \in DL(N)$, let $ET(M,N,l) = \{etree(l,\Gamma_0,\Gamma_1) : \Gamma_0,\Gamma_1 \in ET(M,N(l,\delta)), \delta = 0,1\}$. The next statement follows immediately from the definition of element partition tree.

**Proposition 2.** Let $M$ be a mesh and $N$ be a submesh of $M$. Then $ET(M,N) =$
\{etree(\varphi(N))\} if \(N\) is unitary, and \(ET(M,N) = \bigcup_{l \in DL(N)} ET(M,N,l)\) if \(N\) is nonunitary.

### 2.2.4 Cost Functions for Element Partition Trees

In this section, we define the notion of a cost function for element partition trees.

Each cost function \(\psi\) has values from the set \(\mathbb{R}\) of real numbers and is defined on triples \((M,N,\Gamma)\) where \(M\) is a mesh, \(N\) is a submesh of \(M\), and \(\Gamma\) is an element partition tree for \(N\). The function \(\psi\) is specified by three functions \(\psi^0\), \(F\) and \(w\). Such cost functions are defined inductively as follows.

If \(N\) is a unitary submesh of \(M\) and \(\Gamma = etree(\varphi(N))\), then \(\psi(M,N,\Gamma) = \psi^0(M,N)\) where \(\psi^0\) is a function which is defined on pairs \((M,N)\), where \(M\) is a mesh and \(N\) is a unitary submesh of \(M\), and has values from the set \(\omega = \{0, 1, 2, \ldots\}\) of nonnegative integers.

Let \(N\) be a nonunitary mesh and \(\Gamma = etree(l,\Gamma_0,\Gamma_1)\) where \(l \in DL(N)\), \(\Gamma_\delta \in ET(M,N(l,\delta))\), \(\delta \in \{0, 1\}\). Then

\[
\psi(M,N,\Gamma) = F(\psi(M,N(l,0),\Gamma_0),\psi(M,N(l,1),\Gamma_1)) + w(M,N,l)
\]

where \(F\) is a function which is defined on \(\omega^2\) and has values in \(\omega\) while \(w\) is a function that maps a triplet \((M,N,l)\) to \(\omega\) where \(M\) is a mesh, \(N\) is a submesh of \(M\) and \(l \in DL(N)\).

Using this inductive definition, we can compute the cost of a given element partition tree beginning from terminal nodes and finishing at the root.

Let \(\leq\) be a partial order on the set \(\omega^2\): \((x_1, x_2) \leq (y_1, y_2)\) if \(x_1 \leq y_1\) and \(x_2 \leq y_2\). A function \(f : \omega^2 \to \omega\) is called increasing if \(f(x) \leq f(y)\) for any \(x, y \in \omega^2\) such that \(x \leq y\). The function \(f\) is called strictly increasing if \(f(x) < f(y)\) for any \(x, y \in \omega^2\) such that \(x \leq y\) and \(x \neq y\). For example, \(\max(x_1, x_2)\) and \(x_1 + x_2\) are
increasing functions, and $x_1 + x_2$ is a strictly increasing function. We consider the cost function $\psi$ to be *increasing* if the function $F$ is an increasing function. Similarly, $\psi$ is considered a *strictly increasing* cost function if $F$ is a strictly increasing one. Each strictly increasing cost function is an increasing cost function.

We study different increasing cost functions that model various aspects of complexities. Each of these cost functions is associated with one of two modes of solvers: ASAP and KB. The mode determines the value of some of the parameters ($r$, $n$, $t$, and $m$) used by the function.

All the discussed cost functions depend on a parameter $p$. Some of these cost functions also depend on a parameter $q$. The parameter $p$ (the polynomial order) is a nonnegative integer which identifies the number of basis functions spread over vertices, edges, and interiors of unitary submeshes (finite elements): one per vertex, $p$ per edge, and $p^2$ per interior. The parameter $q$ is a nonnegative integer that characterizes the time of one memory transfer operation relative to the cost of performing an arithmetic operation. Both parameters $p$ and $q$ does not depend on the mode of the solver but on the considered architecture and problem size.

We use $W(\mu, \rho) = \sum_{i=1}^{\rho} (2\mu - 2i + 1)(\mu - i) = 2\mu^2 \rho - 2\mu \rho^2 - \mu \rho + \frac{2}{3} \rho^3 + \frac{1}{2} \rho^2 - \frac{1}{6} \rho$ as the number of arithmetic floating point operations for partial factorization of a $\mu \times \mu$ matrix with $\rho$ fully assembled rows. It is clear that $W(\mu, \rho) \geq 0$. We assume that to keep results of the partial factorization of a $\mu \times \mu$ matrix we need $\mu^2$ memory. We also assume that we should send $(\mu - \rho)^2$ numbers (corresponding to the Schur complement) to the parent node. Note that $\rho \leq \mu$ and $W(\mu, \rho) \leq 4 \times \mu^3$.

Let $N$ be a unitary submesh of $M$, $\Gamma = etree(\varphi(N))$, and $A'$ be an $n \times n$ matrix with $r$ fully assembled rows corresponding to $N$. Then, during the processing of the submesh $N$, the algorithm makes $W(n, r)$ arithmetic floating point operations for partial factorization of $A'$. We send $(n - r)^2$ numbers corresponding to the Schur complement to the parent node if $N \neq M$ and keep the $n^2$ numbers which result
from factorization. We define a parameter $\sigma \in \{0, 1\}$ to indicate whether the current submesh has a parent or not, i.e., $\sigma$ is set to 1 if and only if $N \neq M$.

Let $N$ be a nonunitary submesh, $\Gamma = etree(l, \Gamma_0, \Gamma_1)$ where $l \in DL(N)$ and $\Gamma_\delta \in ET(M, N(l, \delta))$, $\delta \in \{0, 1\}$, and $A''$ be an $m \times m$ matrix with $t$ fully assembled rows corresponding to $N$. Then, during the processing of the submesh $N$, the algorithm makes $W(m, t)$ arithmetic floating point operations during partial factorization of $A''$, it should send $(m-t)^2$ numbers corresponding to the Schur complement to the parent node if $N \neq M$ and store $m^2$ numbers to keep the factorization.

We present the definitions of several cost functions describing different complexity measures.

- **Sequential time** $time^{1,p,q}_\text{mode}$ is a cost function describing the complexity of a sequential solver following a given element partition tree. For this cost function, $\psi^0(M, N) = \sigma q(n - r)^2 + W(n, r)$, $F(x, y) = x + y$, and $w(M, N, l) = \sigma q(m - t)^2 + W(m, t)$. The considered cost function is strictly increasing.

- **Parallel time** $time^{\infty,p,q}_\text{mode}$ is a cost function modeling the complexity of a parallel solver following a given element partition tree. This solver solves the children of a given node in parallel and merges the solution sequentially. For this cost function, $\psi^0(M, N) = \sigma q(n - r)^2 + W(n, r)$, $F(x, y) = \max(x_1, x_2)$, and $w(M, N, l) = \sigma q(m - t)^2 + W(m, t)$. The considered cost function is increasing.

- **Memory** $memory^p_\text{mode}$ is a cost function that measures the space complexity of the solver. For this cost function, $\psi^0(M, N) = n^2$, $F(x, y) = x_1 + x_2$, and $w(M, N, l) = m^2$. The considered cost function is strictly increasing.

The parameters depending on the mode ($r$, $n$, $t$, and $m$) are defined as follows:
If mode is ASAP, we have:

\[ r = p^2 + BE(M,N) \times p + BV(M,N) , \]  
\[ n = p^2 + 4p + 4 , \]  
\[ t = E(M,l) \times (p + 1) + EP(M,N,l) - 1 , \]

\[ m = \begin{cases} 
(B(M,N) - BE(M,N) + E(M,l))(p + 1) + \\
EP(M,N,l) + \gamma , & 1 \leq BE(M,N) < B(M,N) , \\
(B(M,N) - BE(M,N) + E(M,l))(p + 1) + \\
EP(M,N,l) - 1 , & \text{otherwise} , 
\end{cases} \]  

where \( \gamma \in \{0,1\} \) and \( \gamma = 1 \) if and only if \( N \) has exactly two boundary sides which are parallel.

If mode is KB, we have:

\[ r = p^2 , \]
\[ n = p^2 + 4p + 4 , \]
\[ t = E(M,l) \times (p + 1) - 1 , \]
\[ m = (B(M,N) + E(M,l))(p + 1) - 1 . \]

The parameters \( B(M,N) \), \( BE(M,N) \), \( BV(M,N) \), \( E(M,l) \), \( EP(M,N,l) \) and \( \gamma \) can be computed in polynomial time with respect to \( s(M) \). As a result, all of the aforementioned cost functions can be evaluated in polynomial time with respect to the mesh size \( s(M) \). Since \( F \) is either \( \max(x,y) \) or \( x + y \), the time complexity of \( F \) computation is \( O(1) \). As a result, we have the following statement.
Proposition 3. For each cost function

\[ \psi \in \{ \text{time}^{1,p,q}_{\text{mode}}, \text{time}^{\infty,p,q}_{\text{mode}}, \text{memory}^p_{\text{mode}} : \text{mode} \in \{ \text{KB}, \text{ASAP} \} \} \]

defined by the functions \( \psi^0, F \) and \( w \), \( \psi^0, F \) and \( w \) can be computed in polynomial time depending on the size of the mesh \( s(M) \).

Let \( \psi \in \{ \text{time}^{1,p,q}_{\text{mode}}, \text{time}^{\infty,p,q}_{\text{mode}}, \text{memory}^p_{\text{mode}} : \text{mode} \in \{ \text{KB}, \text{ASAP} \} \} \) and \( \psi \) be defined by the functions \( \psi^0, F \) and \( w \).

It is clear that independent of the mode of the solver, \( 0 \leq r \leq n \) and \( 0 \leq t \leq m \).

In addition, we recall from the definitions of the mesh and line parameters that
\[ 0 \leq BE(M,N) \leq B(M,N), \quad 0 \leq EP(M,N,l) \leq 2, \quad B(M,N) \leq 4(s(M) + 1) \text{ and } E(M,l) \leq s(M) + 1. \]
We use this information to show that there exists an upper bound on the value of \( \psi \) which is polynomial on the size of mesh \( s(M) \) and the parameters \( p \) and \( q \).

Let \( N \) be a submesh of a mesh \( M \) and \( \Gamma \) be an element partition tree for \( N \). If \( N \) is a unitary submesh then \( \Gamma \) consists of one node, and the cost of \( \Gamma \) is nonnegative. Otherwise, we notice that \( w(M,N,l) \) is a nonnegative value for any nonunitary submesh \( N \) of \( M \) and \( l \in DL(N) \). As a result, the cost of an element partition tree \( \Gamma' \) for mesh \( M \) with maximum cost forms an upper bound on the cost of any element partition tree for any submesh \( N \) of \( M \).

We denote

\[ ub(\psi, M) = \max\{\psi(M,M,\Gamma) : \Gamma \in ET(M,M)\}, \]

\[ v'(\psi, M) = \max\{\psi^0(M,N) : N \text{ is a unitary submesh of } M\}. \]
and

\[ w'(\psi, M) = \max \{ w(M, N, l) : N \text{ is a nonunitary submesh of } M, l \in DL(N) \} . \]

Consider \( L_t(M) \) to be the number of unitary submeshes of \( M \) and hence the number of leaves for any element partition tree for \( M \).

**Lemma 4.** Let \( \psi \in \{ \text{time}_{\text{mode}}^{1,p,q}, \text{time}_{\text{mode}}^{\infty,p,q}, \text{memory}_{\text{mode}}^p : \text{mode} \in \{ KB, ASAP \} \} \) and \( M \) be a mesh. Then \( ub(\psi, M) \leq s(M)^2(v'(\psi, M) + w'(\psi, M)) \).

**Proof.** Any element partition tree for the mesh \( M \) has all unitary submeshes of \( M \) as leaves. As a result, any element partition tree for \( M \) has exactly \( L_t(M) \) leaves.

Since the element partition tree is a binary tree whose each node either has zero or two children, the number of internal nodes of any element partition tree for \( M \) is \( L_t(M) - 1 \).

The cost of any element partition tree for \( M \) is at most the sum of the costs of its leaves in addition to the sum of the values \( w(M, N, l) \) computed at each node \( N \). Therefore

\[
ub(\psi, M) \leq L_t(M)v'(\psi, M) + (L_t(M) - 1)w'(\psi, M) \\
\leq L_t(M)(v'(\psi, M) + w'(\psi, M)) .
\]

One can show that there exists at most \( s(M)^2 \) unitary submeshes of the mesh \( M \). Thus

\[
ub(\psi, M) \leq s(M)^2(v'(\psi, M) + w'(\psi, M)) .
\]

\( \square \)

**Lemma 5.** Let \( M \) be a mesh and \( \psi \in \{ \text{time}_{\text{mode}}^{1,p,q}, \text{time}_{\text{mode}}^{\infty,p,q} : \text{mode} \in \{ KB, ASAP \} \} \). Then \( ub(\psi, M) = O(s(M)^4q^4 + s(M)^5p^6) \).
Proof. We know that $v'(ψ, M) = ψ^0(M, N)$ for some unitary submesh $N$ of $M$. Therefore

$$v'(ψ, M) = σq(n - r)^2 + W(n, r).$$

Since $σ ∈ \{0, 1\}$ and $W(n, r) ≤ 4n^3$, we have

$$v'(ψ, M) ≤ qn^2 + 4n^3.$$  

From (2.2) and (2.6), we deduce that $n = p^2 + 4p + 4$, and hence

$$v'(ψ, M) ≤ q(p^2 + 4p + 4)^2 + 4(p^2 + 4p + 4)^3 = q(p + 2)^4 + 4(p + 2)^6.$$

We know that $w'(ψ, M) = w(M, N, l)$ for some nonunitary submesh $N$ of $M$ and $l ∈ DL(N)$. Therefore

$$w'(ψ, M) = σq(m - t)^2 + W(m, t).$$

From (2.4) and (2.8), $m ≤ (B(M, N) + E(M, l))(p + 1) + 3$. In addition, since $0 ≤ B(M, N) ≤ 4s(M)$ and $E(M, l) ≤ s(M)$,

$$m ≤ 5s(M)(p + 1) + 3 ≤ 5s(M)(p + 2).$$

On the other hand,

$$w'(ψ, M) = σq(m - t)^2 + W(m, t)$$

$$≤ qm^2 + 4m^3$$

$$≤ 25qs(M)^2(p + 2)^2 + 500s(M)^3(p + 2)^3.$$
From Lemma 4 it follows that

\[
ub(\psi, M) \leq s(M)^2(v'(\psi, M) + w'(\psi, M))
\]
\[
\leq s(M)^2(q(p + 2)^4 + 4(p + 2)^6 + 25q(p + 2)^2s(M)^2
\]
\[
+ 500s(M)^3(p + 2)^3)
\]
\[
\leq qs(M)^2(p + 2)^4 + 4s(M)^2(p + 2)^6 + 25qs(M)^4(p + 2)^2
\]
\[
+ 500s(M)^5(p + 2)^3.
\]

Therefore \( ub(\psi, M) = O(s(M)^4p^4 + s(M)^5p^6) \).

\[\square\]

**Lemma 6.** Let \( M \) be a mesh and \( \psi \in \{ \text{memory}^p_{\text{mode}} : \text{mode} \in \{KB, ASAP\} \} \). Then \( ub(\psi, M) = O(s(M)^4p^4) \).

**Proof.** As in the proof of Lemma 5, we can show that \( v'(\psi, M) = n^2 \) where \( n = (p + 2)^2 \), and \( w'(\psi, M) = m^2 \) where \( m \leq 5s(M)(p + 2) \). Applying Lemma 4, we obtain that

\[
ub(\psi, M) \leq s(M)^2(v'(\psi, M) + w'(\psi, M))
\]
\[
\leq s(M)^2((p + 2)^4 + 25s(M)^2(p + 2)^2)
\]
\[
= s(M)^2(p + 2)^4 + 25s(M)^4(p + 2)^2.
\]

Therefore \( ub(\psi, M) = O(s(M)^4p^4) \). \[\square\]
Chapter 3

Multi-Stage Optimization of Element Partition Trees

An element partition tree describes an ordering of the rows of a matrix describing a system of linear equations. This ordering is essential for solvers to find a solution for our system. The quality of an element partition tree can be evaluated using different criteria. For example, one may be interested in finding an ordering that allows the solver to solve the system of equations using the fewest number of arithmetic operations. Another criterion may focus on the memory usage of the solver and others can focus on the data transfer and the energy consumption [39].

This chapter presents tools and applications of multi-stage optimization of element partition trees. It starts by describing how the set of element partition trees can be represented compactly in the form of a directed acyclic graph (DAG). It then suggests algorithms to count the number of element partition trees represented by a DAG and to optimize the corresponding set of element partition trees with respect to some criterion. The latter algorithm can be used for multi-stage optimization of element partition trees relative to a sequence of cost functions and for the study of totally optimal element partition trees that are optimal with respect to multiple criteria simultaneously. Finally, we present results of experiments on three different classes of adaptive meshes.
3.1 Directed Acyclic Graph $\Delta(M)$

Let $M$ be a mesh. We describe an algorithm $A_1$ for the construction of a directed acyclic graph $\Delta(M)$ which is used to describe and optimize the element partition trees for $M$. The set of nodes of this graph coincides with the set $\text{SUB}(M)$ of submeshes of the mesh $M$. During each iteration (with the exception of the last one) the algorithm processes one node. It starts with the graph that consists of one node $M$ which is not processed yet and finishes when all nodes of the constructed graph are processed.

Algorithm $A_1$ (construction of DAG $\Delta(M)$)

Input: Mesh $M$.

Output: Directed acyclic graph $\Delta(M)$.

1. Construct the graph that consists of one node $M$ which is not marked as processed.

2. If all nodes of the graph are processed then the work of the algorithm is finished. Return the resulting graph as $\Delta(M)$. Otherwise, choose a node (submesh) $N$ that has not been processed yet.

3. If $N$ is a unitary mesh then mark $N$ as processed and proceed to the step 2.

4. If $N$ is nonunitary then, for each $l \in DL(N)$, draw two edges from the node $N$ (this pair of edges is called $l$-pair) and label these edges with pairs $(l, 0)$ and $(l, 1)$. These edges enter nodes $N(l, 0)$ and $N(l, 1)$, respectively. If some of the nodes $N(l, 0), N(l, 1)$ are not present in the graph then add these nodes to the graph. Mark the node $N$ as processed and proceed to the step 2.

Proposition 7. The algorithm $A_1$ has polynomial time complexity with respect to the size of the input mesh.
Proof. The number of iterations of the algorithm $A_1$ (each iteration includes step 2 and step 3 or 4) is at most $|\text{SUB}(M)| + 1$. By Lemma 1, $|\text{SUB}(M)| \leq s(M)^4$. One can show that the time complexity of each iteration is polynomial with respect to the size of $M$. Therefore the algorithm $A_1$ has polynomial time complexity with respect to the size of the input mesh. 

A node of a directed graph is called terminal if there are no edges starting in this node, and internal otherwise. A node $N$ of the graph $\Delta(M)$ is terminal if and only if $N$ is a unitary mesh. A proper subgraph of the graph $\Delta(M)$ is a graph $G$ obtained from $\Delta(M)$ by removal of some $l$-pairs of edges such that each internal node of $\Delta(M)$ keeps at least one $l$-pair of edges starting from this node. By definition, $\Delta(M)$ is a proper subgraph of $\Delta(M)$. Proper subgraphs of the graph $\Delta(M)$ arise as results of the optimization procedure applied to $\Delta(M)$ or to its proper subgraphs.

Let $G$ be a proper subgraph of the graph $\Delta(M)$. For each internal node $N$ of the graph $G$, we denote by $DL_G(N)$ the set of lines $l$ from $DL(N)$ such that an $l$-pair of edges starts from $N$ in $G$. For each node $N$ of the graph $G$, we define the set $Etree(G, N)$ of elimination trees in the following way. If $N$ is a terminal node of $G$ (in this case $N$ is unitary), then $Etree(G, N) = \{etree(\varphi(N))\}$. Let $N$ be an internal node of $G$ (in this case $N$ is nonunitary) and $l \in DL_G(N)$. We denote $Etree(G, N, l) = \{etree(l, \Gamma_0, \Gamma_1) : \Gamma_\delta \in Etree(G, N(l, \delta)), \delta = 0, 1\}$. Then $Etree(G, N) = \bigcup_{l \in DL_G(N)} Etree(G, N, l)$.

Proposition 8. Let $M$ be a mesh. Then the equality

$$Etree(\Delta(M), N) = ET(M, N)$$

holds for any node $N$ of the graph $\Delta(M)$.

Proof. We prove this statement by induction on the nodes of $\Delta(M)$. Let $N$ be a terminal node of $\Delta(M)$. Then $Etree(\Delta(M), N) = \{etree(\varphi(N))\} = ET(M, N)$. On
the other hand, consider $N$ to be an internal node of $\Delta(M)$, and let us assume that
$Etree(\Delta(M), N(l, \delta)) = ET(M, N(l, \delta))$ for any $l \in DL(N)$ and $\delta \in \{0, 1\}$. Then,
for any $l \in DL(N)$, we have $Etree(\Delta(M), N, l) = ET(M, N, l)$, where $DL_{\Delta(M)}(N) = DL(N)$. Using Proposition 2, we obtain $Etree(\Delta(M), N) = ET(M, N)$. \hfill \Box$

So the set of element partition trees for a mesh $M$ is equal to the set

$$Etree(\Delta(M), M).$$

We show later that the set of strictly optimal element partition trees for a mesh $M$
relative to an increasing cost function $\psi$ can be represented in the form $Etree(G, M)$
where $G$ is a proper subgraph of the graph $\Delta(M)$. In the next section, we describe an
algorithm which counts the cardinality of the set $Etree(G, M)$ for a proper subgraph
$G$ of the graph $\Delta(M)$.

### 3.2 Cardinality of the Set $Etree(G, M)$

Let $M$ be a mesh, and $G$ be a proper subgraph of the graph $\Delta(M)$. We describe an
algorithm which counts, for each node $N$ of the graph $G$, the cardinality $C(N)$ of the
set $Etree(G, N)$, and returns the number $C(M) = |Etree(G, M)|$.

*Algorithm $A_2$ (counting the number of element partition trees)*

*Input*: A proper subgraph $G$ of the graph $\Delta(M)$ for some mesh $M$.

*Output*: The number $|Etree(G, M)|$.

1. If all nodes of the graph $G$ are processed then return the number $C(M)$ and
   finish the work of the algorithm. Otherwise, choose a node $N$ which is not
   processed yet and which is either a terminal node of $G$ or an internal node of $G$
such that all nodes $N(l, \delta)$ are processed for each $l \in DL_G(N)$ and $\delta \in \{0, 1\}$. 
2. If $N$ is a terminal node (unitary mesh), set $C(N) = 1$, mark $N$ as processed and proceed to the step 1.

3. If $N$ is an internal node (nonunitary mesh) then set

$$C(N) = \sum_{l \in DL_G(N)} C(N(l,0)) \times C(N(l,1)),$$

mark the node $N$ as processed and proceed to the step 1.

**Proposition 9.** Let $M$ be a mesh, and $G$ be a proper subgraph of the graph $\Delta(M)$. Then the algorithm $A_2$ returns the cardinality of the set $Etree(G, M)$ and makes at most $2s(M)^5$ operations of multiplication and addition.

**Proof.** We prove by induction on the nodes of the graph $G$ that $C(N) = |Etree(G, N)|$ for each node $N$ of $G$. Let $N$ be a terminal node of $G$. Then $Etree(G, N) = \{etree(\varphi(N))\}$ and $|Etree(G, N)| = 1$. Therefore the considered statement holds for $N$. Let now $N$ be an internal node of $G$ such that the considered statement holds for its children. We know that $Etree(G, N) = \bigcup_{l \in DL_G(N)} Etree(G, N, l)$ where, for $l \in DL_G(N),$

$$Etree(G, N, l) = \{etree(l, \Gamma_0, \Gamma_1) : \Gamma_\delta \in Etree(G, N(l, \delta)), \delta = 0, 1\}.$$

Then, for any $l \in DL_G(N),$

$$|Etree(G, N, l)| = |Etree(G, N(l,0))| \times |Etree(G, N(l, 1))|,$$

and $|Etree(G, N)| = \sum_{l \in DL_G(N)} |Etree(G, N, l)|$. By the inductive hypothesis,

$$|Etree(G, N(l, \delta))| = C(N(l, \delta))$$
for each \( l \in DL_G(N) \) and \( \delta \in \{0, 1\} \). Therefore \( C(N) = |Etree(G, N)| \). Hence the considered statement holds. From here it follows that

\[
C(M) = |Etree(G, M)|,
\]

i.e., the algorithm \( A_2 \) returns the cardinality of the set \( Etree(G, M) \).

We evaluate the number of arithmetic operations made by the algorithm \( A_2 \). By Lemma 1, \(|\text{SUB}(M)| \leq s(M)^4\). Therefore the number of internal nodes in \( G \) is at most \( s(M)^4 \). In each internal node \( N \) of \( G \), the algorithm \( A_2 \) makes \( |DL_G(N)| \) operations of multiplication and \( |DL_G(N)| - 1 \) operations of addition. One can show that \( |DL_G(N)| \leq s(M) \). Therefore, the algorithm \( A_2 \) makes at most \( 2s(M)^5 \) arithmetic operations.

### 3.3 Optimization Procedure

Let \( M \) be a mesh, \( G \) be a proper subgraph of the graph \( \Delta(M) \), and \( \psi \) be an increasing cost function for element partition trees defined by functions \( \psi^0, F \) and \( w \).

Let \( N \) be a node of \( G \) and \( \Gamma \in Etree(G, N) \). One can show that, for any node \( v \) of \( \Gamma \), the element partition tree \( \Gamma(v) \) belongs to the set \( Etree(G, N_{\Gamma}(v)) \).

An element partition tree \( \Gamma \) from \( Etree(G, N) \) is called an optimal element partition tree for \( N \) relative to \( \psi \) and \( G \) if

\[
\psi(M, N, \Gamma) = \min \{ \psi(M, N, \Gamma') : \Gamma' \in Etree(G, N) \}.
\]

An element partition tree \( \Gamma \) from \( Etree(G, N) \) is called a strictly optimal element partition tree for \( N \) relative to \( \psi \) and \( G \) if, for any node \( v \) of \( \Gamma \), the element partition tree \( \Gamma(v) \) is an optimal element partition tree for \( N_{\Gamma}(v) \) relative to \( \psi \) and \( G \).

We denote by \( Etree_{\psi}^{opt}(G, N) \) the set of optimal element partition trees for \( N \) relative to \( \psi \) and \( G \). We denote by \( Etree_{\psi}^{s-opt}(G, N) \) the set of strictly optimal
element partition trees for $N$ relative to $\psi$ and $G$.

Let $\Gamma \in Etree_{\psi}^{opt}(G, N)$ and $\Gamma = etree(l, \Gamma_0, \Gamma_1)$. Then $\Gamma \in Etree_{\psi}^{s-opt}(G, N)$ if and only if $\Gamma_\delta \in Etree_{\psi}^{s-opt}(G, N(l, \delta))$ for $\delta = 0, 1$.

**Proposition 10.** Let $\psi$ be a strictly increasing cost function for element partition trees, $M$ be a mesh, and $G$ be a proper subgraph of the graph $\Delta(M)$. Then, for any node $N$ of the graph $G$, $Etree_{\psi}^{opt}(G, N) = Etree_{\psi}^{s-opt}(G, N)$.

**Proof.** It is clear that $Etree_{\psi}^{s-opt}(G, N) \subseteq Etree_{\psi}^{opt}(G, N)$. Let $\Gamma \in Etree_{\psi}^{opt}(G, N)$ and assume that $\Gamma \notin Etree_{\psi}^{s-opt}(G, N)$. Then there is a node $v$ of $\Gamma$ such that $\Gamma(v) \notin Etree_{\psi}^{opt}(G, N(v))$. Let $\Gamma_0 \in Etree_{\psi}^{opt}(G, N(v))$ and $\Gamma'$ be the element partition tree obtained from $\Gamma$ by replacement of $\Gamma(v)$ with $\Gamma_0$. One can show that $\Gamma' \in Etree(G, N)$. Since $\psi$ is strictly increasing and

$$\psi(M, N_\Gamma(v), \Gamma_0) < \psi(M, N_\Gamma(v), \Gamma(v)),$$

we have $\psi(M, N, \Gamma') < \psi(M, N, \Gamma)$. Therefore $\Gamma \notin Etree_{\psi}^{opt}(G, N)$, but this is impossible. Thus $Etree_{\psi}^{opt}(G, N) \subseteq Etree_{\psi}^{s-opt}(G, N)$. \qed

We present an algorithm $\mathcal{A}_3$ which is an optimization procedure relative to a cost function $\psi$. This algorithm attaches to each node $N$ of $G$ the number $c(N) = \min\{\psi(M, N, \Gamma) : \Gamma \in Etree(G, N)\}$ and, probably, removes some $l$-pairs of edges starting from internal nodes of $G$. As a result, we obtain a proper subgraph $G^\psi$ of the graph $G$. By construction, $G^\psi$ is also a proper subgraph of the graph $\Delta(M)$.

**Algorithm $\mathcal{A}_3$** (procedure of element partition tree optimization)

**Input:** A proper subgraph $G$ of the graph $\Delta(M)$ for some mesh $M$, and an increasing cost function $\psi$ for element partition trees defined by functions $\psi^0$, $F$ and $w$.

**Output:** Proper subgraph $G^\psi$ of the graph $G$. 


1. If all nodes of the graph \( G \) are processed then return the obtained graph as \( G^\psi \) and finish. Otherwise, choose a node \( N \) which is not processed yet and which is either a terminal node of \( G \) or an internal node of \( G \) for which all its children have been processed.

2. If \( N \) is a terminal node (unitary mesh) then set \( c(N) = \psi^0(M, N) \), mark \( N \) as processed and proceed to the step 1.

3. If \( N \) is an internal node (nonunitary mesh) then, for each \( l \in DL_G(N) \), compute the value

\[
c(N, l) = F(c(N(l, 0)), c(N(l, 1))) + w(M, N, l)
\]

and set \( c(N) = \min\{c(N, l) : l \in DL_G(N)\} \). Remove all \( l \)-pairs of edges starting from \( N \) for which \( c(N) < c(N, l) \). Mark the node \( N \) as processed and proceed to the step 1.

**Proposition 11.** Let the algorithm \( A_3 \) use a cost function \( \psi \) specified by functions \( \psi^0 \), \( F \) and \( w \) which have polynomial time complexity with respect to the size of the input mesh. Then the algorithm \( A_3 \) has polynomial time complexity with respect to the size of the input mesh.

**Proof.** By Lemma 1, \( |\text{SUB}(M)| \leq s(M)^4 \). Therefore the number of nodes in \( G \) is at most \( s(M)^4 \). In each terminal node of the graph \( G \), the algorithm \( A_3 \) computes the value of \( \psi^0 \). In each internal node \( N \) of \( G \) the algorithm \( A_3 \) computes \( |DL_G(N)| \) times the value of \( F \) and \( w \) and makes \( |DL_G(N)| - 1 \) comparisons. One can show that \( |DL_G(N)| \leq s(M) \). Therefore, the algorithm \( A_3 \) makes at most \( s(M)^5 \) comparisons and at most \( s(M)^5 \) computations of the functions \( \psi^0 \), \( F \) and \( w \). If \( \psi^0 \), \( F \), and \( w \) have polynomial time complexity with respect to \( s(M) \) then the algorithm \( A_3 \) has polynomial time complexity with respect to \( s(M) \). \( \square \)
For any node $N$ of the graph $G$ and for any $l \in DL_G(N)$ we denote $\psi_G(N) = \min\{\psi(M, N, \Gamma) : \Gamma \in Etree(G, N)\}$ and

$$\psi_G(N, l) = \min\{\psi(M, N, \Gamma) : \Gamma \in Etree(G, N, l)\}.$$

**Lemma 12.** Let $\psi$ be an increasing cost function for element partition trees specified by functions $\psi^0$, $F$ and $w$, $M$ be a mesh, and $G$ be a proper subgraph of the graph $\Delta(M)$. Then, for any node $N$ of the graph $G$ and for any $l \in DL_G(N)$, the algorithm $A_3$ computes values $c(N) = \psi_G(N)$ and $c(N, l) = \psi_G(N, l)$.

**Proof.** We prove the considered statement by induction on the nodes of the graph $G$. Let $N$ be a terminal node of $G$. Then $Etree(G, N) = \{etree(\varphi(N))\}$ and $\psi_G(N) = \psi^0(M, N)$. Therefore $c(N) = \psi_G(N)$. Let $N$ be an internal node of $G$ for which the considered statement holds for each node $N(l, \delta)$ such that $l \in DL_G(N)$ and $\delta \in \{0, 1\}$. We know that

$$Etree(G, N) = \bigcup_{l \in DL_G(N)} Etree(G, N, l)$$

and, for each $l \in DL_G(N),$

$$Etree(G, N, l) = \{etree(l, \Gamma_0, \Gamma_1) : \Gamma_\delta \in Etree(G, N(l, \delta)), \delta = 0, 1\}.$$  

Since $\psi$ is an increasing cost function,

$$\psi_G(N, l) = F(\psi_G(N(l, 0)), \psi_G(N(l, 1))) + w(M, N, l).$$

By the induction hypothesis, $\psi_G(N(l, \delta)) = c(N(l, \delta))$ for each $l \in DL_G(N)$ and $\delta \in \{0, 1\}$. Therefore $c(N, l) = \psi_G(N, l)$ for each $l \in DL_G(N)$, and $c(N) = \psi_G(N)$. 

**Theorem 13.** Let $\psi$ be an increasing cost function for element partition trees spec-
ified by functions $\psi^0$, $F$ and $w$, $M$ be a mesh, and $G$ be a proper subgraph of the graph $\Delta(M)$. Then, for any node $N$ of the graph $G^\psi$, the following equality holds:

$$Etree(G^\psi, N) = Etree^{s-opt}_\psi(G, N).$$

Proof. We prove the considered statement by induction on nodes of $G^\psi$. We use Lemma 12 which shows that, for each node $N$ of the graph $G$ and for each $l \in DL_G(N)$, $c(N) = \psi_G(N)$ and $c(N, l) = \psi_G(N, l)$. Let $N$ be a terminal node of $G^\psi$. Then $Etree(G^\psi, N) = \{etree(\varphi(N))\}$ with $Etree(G^\psi, N) = Etree^{s-opt}_\psi(G, N)$.

Let $N$ be an internal node of $G^\psi$. Then

$$Etree(G^\psi, N) = \bigcup_{l \in DL_{G^\psi}(N)} Etree(G^\psi, N, l)$$

and, for each $l \in DL_{G^\psi}(N),

$$Etree(G^\psi, N, l) = \{etree(l, \Gamma_0, \Gamma_1) : \Gamma_\delta \in Etree(G^\psi, N(l, \delta)), \delta = 0, 1\}.$$

Let us assume that, for any $l \in DL_{G^\psi}(N)$ and $\delta \in \{0, 1\},

$$Etree(G^\psi, N(l, \delta)) = Etree^{s-opt}_\psi(G, N(l, \delta)).$$

We know that

$$DL_{G^\psi}(N) = \{l : l \in DL_G(N), \psi_G(N, l) = \psi_G(N)\}.$$

Let $l \in DL_{G^\psi}(N)$, and $\Gamma \in Etree(G^\psi, N, l)$. Then $\Gamma = etree(l, \Gamma_0, \Gamma_1)$, where

$$\Gamma_\delta \in Etree(G^\psi, N(l, \delta))$$
for $\delta = 0, 1$. According to the induction hypothesis,

$$E_{\text{tree}}(G^\psi, N(l, \delta)) = E_{\text{tree}}^{s-\text{opt}}(G, N(l, \delta))$$

and

$$\Gamma_\delta \in E_{\text{tree}}^{s-\text{opt}}(G, N(l, \delta))$$

for $\delta = 0, 1$. In particular, $\psi(M, N(l, \delta), \Gamma_\delta) = \psi_G(N(l, \delta))$ for $\delta = 0, 1$. Since $\psi_G(N, l) = \psi_G(N)$ we have $F(\psi_G(N(l, 0)), \psi_G(N(l, 1))) + w(M, N, l) = \psi_G(N)$. Therefore $\Gamma \in E_{\text{tree}}^{opt}(G, N)$, $\Gamma \in E_{\text{tree}}^{s-\text{opt}}(G, N)$, and

$$E_{\text{tree}}(G^\psi, N) \subseteq E_{\text{tree}}^{s-\text{opt}}(G, N).$$

Let $\Gamma \in E_{\text{tree}}^{s-\text{opt}}(G, N)$. Since $N$ is an internal node, $\Gamma$ can be represented in the form $\text{etree}(l, \Gamma_0, \Gamma_1)$ where $l \in DL_G(N)$, and $\Gamma_\delta \in E_{\text{tree}}^{s-\text{opt}}(G, N(l, \delta))$ for $\delta = 0, 1$. Since $\Gamma \in E_{\text{tree}}^{s-\text{opt}}(G, N)$, $\psi_G(N, l) = \psi_G(N)$ and $l \in DL_{G^\psi}(N)$. According to the induction hypothesis,

$$E_{\text{tree}}(G^\psi, N(l, \delta)) = E_{\text{tree}}^{s-\text{opt}}(G, N(l, \delta))$$

for $\delta = 0, 1$. Therefore $\Gamma \in E_{\text{tree}}(G^\psi, N, l) \subseteq E_{\text{tree}}(G^\psi, N)$. As a result, we have

$$E_{\text{tree}}^{s-\text{opt}}(G, N) \subseteq E_{\text{tree}}(G^\psi, N).$$

\[\square\]

**Corollary 14.** Let $\psi$ be a strictly increasing cost function for element partition trees, $M$ be a mesh, and $G$ be a proper subgraph of the graph $\Delta(M)$. Then, for any node $N$ of $G^\psi$, $E_{\text{tree}}(G^\psi, N) = E_{\text{tree}}^{\text{opt}}(G, N)$.

This corollary follows immediately from Proposition 10 and Theorem 13.
3.4 Multi-Stage Optimization

We use a multi-stage optimization for the element partition trees for $M$ relative to a sequence of strictly increasing cost functions $\psi_1, \ldots, \psi_l$. We begin from the graph $G = \Delta(M)$ and apply to it the procedure of optimization (the algorithm $A_3$) relative to the cost function $\psi_1$. As a result, we obtain a proper subgraph $G^{\psi_1}$ of the graph $G = \Delta(M)$. By Proposition 8, the set $Etree(G, M)$ is equal to the set $ET(M, M)$ of all element partition trees for $M$. Using Corollary 14, we obtain that the set $Etree(G^{\psi_1}, M)$ coincides with the set $Etree_{opt}^{\psi_1}(G, M)$ of all element partition trees from $Etree(G, M)$ which have minimum cost relative to $\psi_1$ among all trees from $Etree(G, M)$. Next we apply to $G^{\psi_1}$ the procedure of optimization relative to the cost function $\psi_2$. As a result, we obtain a proper subgraph $G^{\psi_1, \psi_2}$ of the graph $G^{\psi_1}$ (and of the graph $G = \Delta(M)$). By Corollary 14, the set $Etree(G^{\psi_1, \psi_2}, M)$ coincides with the set $Etree_{opt}^{\psi_2}(G^{\psi_1}, M)$ of all element partition trees from $Etree(G^{\psi_1}, M)$ which have minimum cost relative to $\psi_2$ among all trees from $Etree(G^{\psi_1}, M)$, etc.

If one of the cost functions $\psi_i$ is not strictly increasing but increasing only, then the set $Etree(G^{\psi_1, \ldots, \psi_i}, M)$ coincides with the set $Etree_{opt}^{s-opt}(G^{\psi_1, \ldots, \psi_{i-1}}, M)$ which is a subset of the set of all element partition trees from $Etree(G^{\psi_1, \ldots, \psi_{i-1}}, M)$ that have minimum cost relative to $\psi_i$ among all trees from $Etree(G^{\psi_1, \ldots, \psi_{i-1}}, M)$.

3.5 Totally Optimal Trees

Let $M$ be a mesh. For a cost function $\psi$, we denote $\psi(M) = \min\{\psi(M, M, \Gamma) : \Gamma \in ET(M, M)\}$, i.e., $\psi(M)$ is the minimum cost of an element partition tree for $M$ relative to the cost function $\psi$. An element partition tree $\Gamma$ for $M$ is called $totally$ $optimal$ $relative$ $to$ $cost$ $functions$ $\psi_1$ $and$ $\psi_2$ if $\psi_1(M, M, \Gamma) = \psi_1(M)$ and $\psi_2(M, M, \Gamma) = \psi_2(M)$, i.e., $\Gamma$ is optimal relative to $\psi_1$ and $\psi_2$ simultaneously.

Let us assume $\psi_1$ is strictly increasing and $\psi_2$ is increasing or strictly increasing.
We now describe how to recognize the existence of an element partition tree for \( M \) which is totally optimal relative to cost functions \( \psi_1 \) and \( \psi_2 \).

Let \( G = \Delta(M) \). First, we apply to \( G \) the procedure of optimization relative to \( \psi_1 \). As a result, we obtain the graph \( G^{\psi_1} \) and the number \( \psi_1(M) \) attached to the node \( M \) of \( G^{\psi_1} \). Next, we apply to \( G \) the procedure of optimization relative to \( \psi_2 \). As a result, we obtain the graph \( G^{\psi_2} \) and the number \( \psi_2(M) \) attached to the node \( M \) of \( G^{\psi_2} \). After that, we apply to \( G^{\psi_1} \) the procedure of optimization relative to \( \psi_2 \). As a result, we obtain the graph \( G^{\psi_1,\psi_2} \) and the number \( \min \{ \psi_2(M, M, \Gamma) : \Gamma \in Etree(G^{\psi_1}, M) \} \) attached to the node \( M \) of \( G^{\psi_1,\psi_2} \). One can show that a totally optimal relative to \( \psi_1 \) and \( \psi_2 \) element partition tree for \( M \) exists if and only if \( \min \{ \psi_2(M, M, \Gamma) : \Gamma \in Etree(G^{\psi_1}, M) \} = \psi_2(M) \).

In some cases, totally optimal trees do not exist and what we can seek is to find optimal trees under some constraints. For example, most of the time we are looking for a tree which is optimal with respect to time complexity, but when the problem becomes large enough, the solver may fail due to the memory constraints. At this point, we can use multi-stage optimization to find fastest trees among those that respect the memory constraints. Thus, we can determine trees that are fastest within trees with the least memory needs. These trees may not be totally optimal trees, however they may answer important practical questions.

### 3.6 Experimental Study of Three Types of Meshes

In this section, we consider results of experimental study of optimal and totally optimal element partition trees for three types of meshes.

#### 3.6.1 Experimental Settings

For each solver mode (ASAP and KB) we use five specific cases of the cost functions described in Section 2.2.4. Two cost functions characterize the number of arith-
metic floating point operations for solvers working in single-processor environment \((time_{mode}^{1,p,0})\) and multi-processor environment \((time_{mode}^{\infty,p,0})\). It is known that memory transfer operations can be more expensive than arithmetic operations. So we consider two other cost functions counting memory transfer operations in addition to arithmetic operations in single-processor environment \((time_{mode}^{1,p,107})\) and multi-processor environment \((time_{mode}^{\infty,p,107})\). We assume, based on some experimental work with GPU computations, that a memory transfer operation is as expensive as 107 arithmetic operations. The last cost function \((memory_{mode}^{p})\) measures the memory consumption of solver following a given element partition tree. All cost functions under consideration are strictly increasing except \(time_{mode}^{\infty,p,0}\) and \(time_{mode}^{\infty,p,107}\) that are only increasing.

A finite element mesh may be refined in some regions where the relative error of approximation is higher as done in the adaptive finite element method [40, 41]. We consider a refinement step as the step where all elements nearest to the singularity position are divided into four finite elements. This refinement step improves the quality of approximation however it increases the complexity of computing the approximation as the mesh size increases.

We consider three types of meshes that differ according to the singularity type. All of them start with a mesh that contains only one element, and a number of refinement steps \(k\) is specified and performed. The first class of meshes (Point-singular mesh) has the bottom right corner as the position of the singularity. The second class (Edge-singular mesh) has the bottom edge as the singularity position. While the last class (Angle-singular mesh) has its bottom and right edges as positions of singularity in addition to their intersection point. Figs. 3.1, 3.2 and 3.3 present schematic examples of each of the previous classes of meshes. Here \(P_k\), \(E_k\) and \(A_k\) refer to a Point-singular mesh, Edge-singular mesh, and Angle-singular mesh with \(k\) refinement steps, respectively.

Another parameter that affects the complexity of the element partition tree opti-
mization problem is the polynomial order $p$. We specify a given optimization problem by the mesh type, the number of refinement steps $k$, and the polynomial order $p$ which is one of the parameters of the considered cost functions. For each type of mesh, we work with $1 \leq k \leq 10$ refinements and $1 \leq p \leq 5$ polynomial orders, i.e., with 50 optimization problems.

### 3.6.2 Optimal Element Partition Trees

This section shows the results of optimizing the element partition trees with respect to a single cost function. We find the number of optimal element partition trees for all meshes and cost functions under consideration with the exception of the cost functions $time_{mode}^{\infty,p,0}$ and $time_{mode}^{\infty,p,107}$. The aforementioned cost functions are increasing so our algorithm can describe only a subset of the set of optimal element partition trees for those cost functions, more precisely, the set of strictly optimal element partition trees.
As the number of results is too large to include here, we present only the results associated with $time_{mode}^{1,p,107}$ cost functions for the studied mesh classes. The column “# all trees” in Tables 3.1, 3.3, and 3.5 contains the number of all element partition trees for the meshes $P_k$, $E_k$, and $A_k$, respectively. This number does not depend on the mode ASAP or KB. The number of trees increases too fast with the number of refinement steps. For example, the number of element partition trees for $A_{10}$ is more than $10^{1603}$. This prevents a naive search algorithm from solving the problem.

Tables 3.1 and 3.2 summarize the number of optimal elimination trees for $P_k$ meshes with different polynomial orders $p$ for KB and ASAP solvers, respectively. The $P_k$ meshes are simpler than the other investigated classes so we are able to understand the behavior of any such optimal trees.

The optimal tree for $time_{ASAP}^{1,p,107}$ can be described in a recursive manner. For the $P_1$ mesh, there are only two steps possible and their order does not matter. For a general $P_k$ mesh, the optimal strategy uses a pair of steps based on vertical and horizontal middle lines in any order (e.g, $AB$ and $CD$ in Figure 3.1). It then obtains three unitary submeshes that are not further processed and a $P_{k-1}$ submesh which is partitioned using the same strategy. Since going from a $P_k$ mesh to a $P_{k-1}$ submesh, the tree takes two steps in a non-deterministic order then the number of optimal trees for $P_k$ is $2^k$ as computed by our algorithm.

An optimal tree for $time_{KB}^{1,p,107}$ and a $P_k$ mesh is harder to describe. For a $P_1$ mesh,
we have the same optimal trees as before. For a general $P_k$ mesh, the optimal strategy chooses at the beginning one of the horizontal and vertical middle lines and then it continues deterministically until it obtains a $P_1$ mesh. The last $P_1$ mesh obtained is dealt with as described above. The number of optimal trees is four because initially there are two possible first steps and two possible pairs of last steps (e.g., $(EF, GH)$ and $(GH, EF)$ in Fig. 3.1).

Tables 3.3 and 3.4 depict the number of optimal element partition trees for $E_k$ meshes. There is exactly one optimal element partition tree for $k > 1$ however those element partition trees may change while varying $p$. Moreover, the optimal element partition tree for an $E_k$ mesh and $time^{1,p,107}_{KB}$ is different from that for an $E_k$ mesh and $time^{1,p,107}_{ASAP}$.

For $A_k$ meshes, the situation is more complicated. Tables 3.5 and 3.6 present the number of optimal trees for an $A_k$ mesh and cost functions $time^{1,p,107}_{KB}$ and $time^{1,p,107}_{ASAP}$, respectively. We are not able to find a clear pattern describing the optimal strategy for such meshes.

Table 3.1: Number of optimal element partition trees for mesh $P_k$ and cost function $time^{1,p,107}_{KB}$

<table>
<thead>
<tr>
<th>$k$</th>
<th># all trees</th>
<th>$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>54</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>162</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>486</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1458</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4374</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>13122</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>39366</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.1: Number of optimal element partition trees for mesh $P_k$ and cost function $time^{1,p,107}_{KB}$
Table 3.2: Number of optimal element partition trees for mesh $P_k$ and cost function $time_{ASAP}$

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
</tr>
</tbody>
</table>

3.6.3 Totally Optimal Element Partition Trees

This section contains experimental results for totally optimal element partition trees. The summary of the results is found in Tables 3.7, 3.9 and 3.11. For each type of mesh and each of the considered eight pairs of cost functions these tables contain a number of optimization problems (pairs $k$ and $p$) for which a totally optimal tree relative to the considered pair of cost functions exists. As we mentioned before, the whole number of optimization problems (pairs $k$ and $p$) is equal to 50.

For a $P_k$ mesh, we find in many cases totally optimal element partition trees for all of the considered problems (see Table 3.7). We focus on the pair of cost functions $(time_{KB}^{1,p,107},time_{KB}^{\infty,p,0})$ where we have totally optimal element partition trees for 42 from 50 cases (see Table 3.8 where 1 means existence and 0 means non-existence of a totally optimal tree). There exist totally optimal trees for all the optimization problems other than those where $p = 1$ and $k > 2$.

For an $E_k$ mesh, we find totally optimal element partition trees for all 50 problems only for $memory_{KB}^{p}$ and $time_{KB}^{1,p,107}$ (see Table 3.9). We highlight a case where totally optimal trees exist for 23 of the 50 problems in Table 3.10. This is noticed with the pair of cost functions $(time_{KB}^{1,p,107},time_{KB}^{\infty,p,107})$. 
Finally, for $A_k$ meshes, there does not exist a pair of cost functions that have totally optimal elimination trees for all 50 problems (see Table 3.11). However, the pair of cost functions $(\text{memory}^{p}_{KB}, \text{time}^{1,p,107}_{KB})$ have totally optimal trees for 38 of the investigated 50 problems. We present which problems have totally optimal element partition trees for the pair $(\text{time}^{1,p,107}_{KB}, \text{time}^\infty_{KB})$ in Table 3.12.
Table 3.4: Number of optimal element partition trees for mesh $E_k$ and cost function $time^{1,p,107}_{ASAP}$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$p$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.5: Number of optimal element partition trees for mesh $A_k$ and cost function $time^{1,p,107}_{KB}$.

<table>
<thead>
<tr>
<th>$k$</th>
<th># all trees</th>
<th>$p$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>580</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>$2.6 \times 10^8$</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$2.9 \times 10^{20}$</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$1 \times 10^{45}$</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$6.6 \times 10^{94}$</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$9.2 \times 10^{194}$</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$6.5 \times 10^{395}$</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>$1.2 \times 10^{798}$</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$1.7 \times 10^{1603}$</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>$k$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>-----</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>9</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>10</td>
<td>512</td>
<td>512</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
</tr>
</tbody>
</table>

Table 3.6: Number of optimal element partition trees for mesh $A_k$ and cost function $time_{ASAP}^{1,p,107}$

<table>
<thead>
<tr>
<th>Pair of cost functions</th>
<th>mode</th>
<th>ASAP</th>
<th>KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$memory_{mode}^p$, $time_{mode}^{1,p,0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$memory_{mode}^p$, $time_{mode}^{\infty,p,0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$memory_{mode}^p$, $time_{mode}^{1,p,107}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$memory_{mode}^p$, $time_{mode}^{\infty,p,107}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$time_{mode}^{1,p,0}$, $time_{mode}^{\infty,p,0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$time_{mode}^{1,p,0}$, $time_{mode}^{\infty,p,107}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$time_{mode}^{1,p,107}$, $time_{mode}^{\infty,p,0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$time_{mode}^{1,p,107}$, $time_{mode}^{\infty,p,107}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.7: Existence of totally optimal element partition trees for instances of $P_k$ meshes where $1 \leq k \leq 10$ and $1 \leq p \leq 5$. 
Table 3.8: Existence of totally optimal trees for $P_k$ relative to $time^{1,p,107}_{KB}$, $time^\infty_{KB}$

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.9: Existence of totally optimal element partition trees for instances of $E_k$ meshes where $1 \leq k \leq 10$ and $1 \leq p \leq 5$.

<table>
<thead>
<tr>
<th>Pair of cost functions</th>
<th>mode</th>
<th>ASAP</th>
<th>KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$memory^p_{mode}$, $time^{1,p,0,0}_{mode}$</td>
<td>ASAP</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$memory^p_{mode}$, $time^\infty_{mode}$</td>
<td>ASAP</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>$memory^p_{mode}$, $time^{1,p,107,0}_{mode}$</td>
<td>ASAP</td>
<td>43</td>
<td>50</td>
</tr>
<tr>
<td>$memory^p_{mode}$, $time^\infty_{mode}$</td>
<td>ASAP</td>
<td>10</td>
<td>23</td>
</tr>
<tr>
<td>$time^{1,p,0,0}<em>{mode}$, $time^\infty</em>{mode}$</td>
<td>ASAP</td>
<td>9</td>
<td>31</td>
</tr>
<tr>
<td>$time^{1,p,107,0}<em>{mode}$, $time^\infty</em>{mode}$</td>
<td>ASAP</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>$time^{1,p,107}<em>{mode}$, $time^\infty</em>{mode}$</td>
<td>ASAP</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>$time^{1,p,107}<em>{mode}$, $time^\infty</em>{mode}$</td>
<td>ASAP</td>
<td>10</td>
<td>23</td>
</tr>
</tbody>
</table>
Table 3.10: Existence of totally optimal trees for $E_k$ relative to $time_{KB}^{1,p,107}$, $time_{KB}^{\infty,p,107}$

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.11: Existence of totally optimal element partition trees for instances of $A_k$ meshes where $1 \leq k \leq 10$ and $1 \leq p \leq 5$.

<table>
<thead>
<tr>
<th>Pair of cost functions</th>
<th>mode</th>
<th>ASAP</th>
<th>KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$memory_{mode}^p$, $time_{mode}^{1,p,0}$</td>
<td>ASAP</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$memory_{mode}^p$, $time_{mode}^{\infty,p,0}$</td>
<td>ASAP</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>$memory_{mode}^p$, $time_{mode}^{1,p,107}$</td>
<td>ASAP</td>
<td>36</td>
<td>38</td>
</tr>
<tr>
<td>$memory_{mode}^p$, $time_{mode}^{\infty,p,107}$</td>
<td>ASAP</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>$time_{mode}^{1,p,0}$, $time_{mode}^{\infty,p,0}$</td>
<td>ASAP</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>$time_{mode}^{1,p,0}$, $time_{mode}^{\infty,p,107}$</td>
<td>ASAP</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$time_{mode}^{1,p,107}$, $time_{mode}^{\infty,p,0}$</td>
<td>ASAP</td>
<td>10</td>
<td>17</td>
</tr>
<tr>
<td>$time_{mode}^{1,p,107}$, $time_{mode}^{\infty,p,107}$</td>
<td>ASAP</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>
Table 3.12: Existence of totally optimal trees for $A_k$ relative to $time_{KB}^{1,p,107}$, $time_{KB}^{\infty,p,0}$

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Chapter 4

Bi-Criteria Optimization of Element Partition Trees

Sometimes it is important to take into consideration two or more criteria when evaluating the quality of an element partition tree. Moreover, the relationship between different criteria is interesting for different problems. For example, the minimization of the number of arithmetic operations is shown to be different than minimizing the fill-in in [42]. Another example focus on examining the relationships between improving the runtime of compiler performance and the energy consumption as studied in [43].

In the previous chapter, we discussed totally optimal element partition trees. Those are trees that are optimal with respect to multiple criteria simultaneously. For example, we might have element partition trees that are optimal from the point of view of both time and memory complexity. However, it is not a common case to have such trees. Thus, it becomes important to find Pareto optimal element partition trees with respect to different criteria.

This chapter presents an algorithm for finding Pareto optimal points corresponding to Pareto optimal element partition trees, and some applications of this algorithm. An additional application of improving the runtime of iterative solver is discussed in [9].

Results mentioned in Section 4.1 are obtained as a joint work with my colleagues: Shahid Hussain, Talha Amin, Mohammad Azad and Fawaz Alsolami.
4.1 Tools

We consider tools (statements and algorithms) which are used for the study of Pareto optimal points for bi-criteria optimization problems. The term Pareto optimal point is named after Vilfredo Pareto (1848–1923).

We correspond to each element partition tree a point whose coordinates are values of the two considered criteria for the given tree. We study the problem of minimization of these criteria, and construct for this problem all Pareto optimal points (nondominated points). It is easy to accomplish this task if the number of element partition trees under consideration is reasonable. However, if the number of element partition trees is huge (as in our case), we propose an algorithm which is based on the tools considered in this section.

Let \( \mathbb{R}^2 \) be the set of pairs of real numbers (points). We consider a partial order \( \leq \) on the set \( \mathbb{R}^2 \) (on the plane): \( (c,d) \leq (a,b) \) if \( c \leq a \) and \( d \leq b \). Two points \( \alpha \) and \( \beta \) are comparable if \( \alpha \leq \beta \) or \( \beta \leq \alpha \). A subset of \( \mathbb{R}^2 \) in which no two different points are comparable is called an antichain. We will write \( \alpha < \beta \) if \( \alpha \leq \beta \) and \( \alpha \neq \beta \). If \( \alpha \) and \( \beta \) are comparable then \( \min(\alpha, \beta) = \alpha \) if \( \alpha \leq \beta \) and \( \min(\alpha, \beta) = \beta \) if \( \alpha > \beta \).

Let \( A \) be a nonempty finite subset of \( \mathbb{R}^2 \). A point \( \alpha \in A \) is called a Pareto optimal point for \( A \) if there is no a point \( \beta \in A \) such that \( \beta < \alpha \). We denote by \( \text{Par}(A) \) the set of Pareto optimal points for \( A \). It is clear that \( \text{Par}(A) \) is an antichain. Note that the considered notations are not unique in the literature [44,45].

**Lemma 15.** Let \( A \) be a nonempty finite subset of the set \( \mathbb{R}^2 \). Then, for any point \( \alpha \in A \), there is a point \( \beta \in \text{Par}(A) \) such that \( \beta \leq \alpha \).

**Proof.** Let \( \beta = (a,b) \) be a point from \( A \) such that \( (a,b) \leq \alpha \) and \( a + b = \min\{c + d : (c,d) \in A, (c,d) \leq \alpha \} \). Then \( (a,b) \in \text{Par}(A) \). \( \square \)
**Lemma 16.** Let $A$ be a nonempty finite subset of $\mathbb{R}^2$. Then

$$|\text{Par}(A)| \leq \min \left( |A^{(1)}|, |A^{(2)}| \right)$$

where $A^{(1)} = \{a : (a, b) \in A\}$ and $A^{(2)} = \{b : (a, b) \in A\}$.

**Proof.** Let $(a, b), (c, d) \in \text{Par}(A)$ and $(a, b) \neq (c, d)$. Then $a \neq c$ and $b \neq d$ (otherwise, $(a, b)$ and $(c, d)$ are comparable). Therefore $|\text{Par}(A)| \leq \min \left( |A^{(1)}|, |A^{(2)}| \right)$. \qed

Points from $\text{Par}(A)$ can be ordered in the following way: $(a_1, b_1), \ldots, (a_t, b_t)$ where $a_1 < \ldots < a_t$. Since points from $\text{Par}(A)$ are incomparable, $b_1 > \ldots > b_t$. We will refer to the sequence $(a_1, b_1), \ldots, (a_t, b_t)$ as the normal representation of the set $\text{Par}(A)$.

It is well known (see [46]) that there exists an algorithm which, for a given set $A$, constructs the set $\text{Par}(A)$ and makes $O(|A| \log |A|)$ comparisons.

For the sake of completeness, we describe an algorithm which, for a given nonempty finite subset $A$ of the set $\mathbb{R}^2$, constructs the normal representation of the set $\text{Par}(A)$. We assume that $A$ is a multiset containing, possibly, repeating elements. The cardinality $|A|$ of $A$ is the total number of elements in $A$.

**Algorithm $A_4$** (construction of normal representation for the set of POPs)

*Input:* A nonempty finite subset $A$ of the set $\mathbb{R}^2$ containing, possibly, repeating elements (multiset).

*Output:* Normal representation $P$ of the set $\text{Par}(A)$ of Pareto optimal points for $A$.

1. Set $P$ equal to the empty sequence.

2. Construct a sequence $B$ of all points from $A$ ordered according to the first coordinate in the ascending order.
3. If there is only one point in the sequence $B$, then add this point to the end of the sequence $P$, return $P$, and finish the work of the algorithm. Otherwise, choose the first $\alpha = (\alpha_1, \alpha_2)$ and the second $\beta = (\beta_1, \beta_2)$ points from $B$.

4. If $\alpha$ and $\beta$ are comparable then remove $\alpha$ and $\beta$ from $B$, add the point $\min(\alpha, \beta)$ to the beginning of $B$, and proceed to step 3.

5. If $\alpha$ and $\beta$ are not comparable (in this case $\alpha_1 < \beta_1$ and $\alpha_2 > \beta_2$) then remove $\alpha$ from $B$, add the point $\alpha$ to the end of $P$, and proceed to step 3.

**Proposition 17.** Let $A$ be a nonempty finite subset of the set $\mathbb{R}^2$ containing, possibly, repeating elements (multiset). Then the algorithm $A_4$ returns the normal representation of the set $\text{Par}(A)$ of Pareto optimal points for $A$ and makes $O(|A| \log |A|)$ comparisons.

**Proof.** The step 2 of the algorithm requires $O(|A| \log |A|)$ comparisons. Each call to step 3 (with the exception of the last one) leads to two comparisons. The number of calls to step 3 is at most $|A|$. Therefore the algorithm $A_4$ makes $O(|A| \log |A|)$ comparisons.

Let the output sequence $P$ be equal to $(a_1, b_1), \ldots, (a_t, b_t)$ and let us set $Q = \{(a_1, b_1), \ldots, (a_t, b_t)\}$. It is clear that $a_1 < \ldots < a_t$, $b_1 > \ldots > b_t$ and, for any $\alpha \in A$, $\alpha \notin Q$, there exists $\beta \in Q$ such that $\beta < \alpha$. From here it follows that $\text{Par}(A) \subseteq Q$ and $Q$ is an antichain. Let us assume that there exists $\gamma \in Q$ which does not belong to $\text{Par}(A)$. Then there exists $\alpha \in A$ such that $\alpha < \gamma$. Since $Q$ is an antichain, $\alpha \notin Q$. We know that there exists $\beta \in Q$ such that $\beta \leq \alpha$. This results in two different points $\beta$ and $\gamma$ from $Q$ being comparable, which is impossible. Therefore $Q = \text{Par}(A)$ and $P$ is the normal representation of the set $\text{Par}(A)$.

**Lemma 18.** Let $A$ be a nonempty finite subset of $\mathbb{R}^2$, $B \subseteq A$, and $\text{Par}(A) \subseteq B$. Then $\text{Par}(B) = \text{Par}(A)$.
Proof. It is clear that \( \text{Par}(A) \subseteq \text{Par}(B) \). Let us assume that, for some \( \beta, \beta \in \text{Par}(B) \) and \( \beta \notin \text{Par}(A) \). Then there exists \( \alpha \in A \) such that \( \alpha < \beta \). By Lemma 15, there exists \( \gamma \in \text{Par}(A) \subseteq B \) such that \( \gamma \leq \alpha \). Therefore \( \gamma < \beta \) and \( \beta \notin \text{Par}(B) \). Hence \( \text{Par}(B) = \text{Par}(A) \).

Lemma 19. Let \( A_1, \ldots, A_k \) be nonempty finite subsets of \( \mathbb{R}^2 \). Then \( \text{Par}(A_1 \cup \ldots \cup A_k) \subseteq \text{Par}(A_1) \cup \ldots \cup \text{Par}(A_k) \).

Proof. Let \( \alpha \in (A_1 \cup \ldots \cup A_k) \setminus (\text{Par}(A_1) \cup \ldots \cup \text{Par}(A_k)) \). Then there is \( i \in \{1, \ldots, k\} \) such that \( \alpha \in A_i \) but \( \alpha \notin \text{Par}(A_i) \). Therefore there is \( \beta \in A_i \) such that \( \beta < \alpha \). Hence \( \alpha \notin \text{Par}(A_1 \cup \ldots \cup A_k) \), and \( \text{Par}(A_1 \cup \ldots \cup A_k) \subseteq \text{Par}(A_1) \cup \ldots \cup \text{Par}(A_k) \).

A function \( f : \mathbb{R}^2 \to \mathbb{R} \) is called increasing if \( f(x_1, y_1) \leq f(x_2, y_2) \) for any \( x_1, x_2, y_1, y_2 \in \mathbb{R} \) such that \( x_1 \leq x_2 \) and \( y_1 \leq y_2 \). For example, \( \text{sum}(x, y) = x + y \) and \( \text{max}(x, y) \) are increasing functions.

Let \( f, g \) be increasing functions from \( \mathbb{R}^2 \) to \( \mathbb{R} \), and \( A, B \) be nonempty finite subsets of the set \( \mathbb{R}^2 \). We denote by \( A \langle fg \rangle B \) the set \( \{(f(a, c), g(b, d)) : (a, b) \in A, (c, d) \in B\} \).

Lemma 20. Let \( A, B \) be nonempty finite subsets of \( \mathbb{R}^2 \), and \( f, g \) be increasing functions from \( \mathbb{R}^2 \) to \( \mathbb{R} \). Then \( \text{Par}(A \langle fg \rangle B) \subseteq \text{Par}(A) \langle fg \rangle \text{Par}(B) \).

Proof. Let \( \beta \in \text{Par}(A \langle fg \rangle B) \) and \( \beta = (f(a, c), g(b, d)) \) where \( (a, b) \in A \) and \( (c, d) \in B \). Then, by Lemma 15, there exist \( (a', b') \in \text{Par}(A) \) and \( (c', d') \in \text{Par}(B) \) such that \( (a', b') \leq (a, b) \) and \( (c', d') \leq (c, d) \). It is clear that \( \alpha = (f(a', c'), g(b', d')) \leq (f(a, c), g(b, d)) = \beta \), and \( \alpha \in \text{Par}(A) \langle fg \rangle \text{Par}(B) \). Since \( \beta \in \text{Par}(A \langle fg \rangle B) \), we have \( \beta = \alpha \). Therefore \( \text{Par}(A \langle fg \rangle B) \subseteq \text{Par}(A) \langle fg \rangle \text{Par}(B) \).

Let \( A \) be a nonempty finite subset of \( \mathbb{R}^2 \). We correspond to \( A \) a partial function \( \mathcal{F}_A : \mathbb{R} \to \mathbb{R} \) defined in the following way: \( \mathcal{F}_A(x) = \min\{b : (a, b) \in A, a \leq x\} \).
Lemma 21. Let $A$ be a nonempty finite subset of $\mathbb{R}^2$, and $(a_1, b_1), \ldots, (a_t, b_t)$ be the normal representation of the set $\text{Par}(A)$. Then, for any $x \in \mathbb{R}$, $\mathcal{F}_A(x) = \mathcal{F}(x)$ where

$$
\mathcal{F}(x) = \begin{cases} 
\text{undefined}, & x < a_1 \\
 b_1, & a_1 \leq x < a_2 \\
 & \cdots \\
 b_{t-1}, & a_{t-1} \leq x < a_t \\
 b_t, & a_t \leq x
\end{cases}
$$

Proof. One can show that $a_1 = \min\{a : (a, b) \in A\}$. Therefore the value $\mathcal{F}_A(x)$ is undefined if $x < a_1$. Let $x \geq a_1$. Then both values $\mathcal{F}(x)$ and $\mathcal{F}_A(x)$ are defined. It is easy to check that $\mathcal{F}(x) = \mathcal{F}_{\text{Par}(A)}(x)$. Since $\text{Par}(A) \subseteq A$, we have $\mathcal{F}_A(x) \leq \mathcal{F}(x)$. By Lemma 15, for any point $(a, b) \in A$, there is a point $(a_i, b_i) \in \text{Par}(A)$ such that $(a_i, b_i) \leq (a, b)$. Therefore $\mathcal{F}(x) \leq \mathcal{F}_A(x)$ and $\mathcal{F}_A(x) = \mathcal{F}(x)$. \qed

4.2 Bi-Criteria Optimization

In this section, we describe an algorithm that computes the set of Pareto optimal points for bi-criteria optimization problems of element partition trees. In addition, we show a natural way to create a graph describing the relationship between the considered cost functions based on the set of Pareto optimal points found.

Let $F, H$ be increasing functions from $\mathbb{R}^2$ to $\mathbb{R}$, and $A, B$ be nonempty finite subsets of the set $\mathbb{R}^2$. Note that

$$
A\langle FH \rangle B = \{(F(a, c), H(b, d)) : (a, b) \in A, (c, d) \in B\}.
$$

Let $\psi$ and $\lambda$ be increasing cost functions for element partition trees specified by triples of functions $\psi^0, F, w$ and $\lambda^0, H, u$, respectively. Consider $M$ to be a mesh and
\[ G = \Delta (M). \] For each node \( N \) of the graph \( G \), we define the set

\[ s_{\psi,\lambda}(G, N) = \left\{ (\psi(M, N, \Gamma), \lambda(M, N, \Gamma)) : \Gamma \in Etree(G, N) \right\}. \]

Note that, by Proposition 8, the set \( Etree(G, N) \) is equal to \( ET(M, N) \). We denote by \( Par(s_{\psi,\lambda}(G, N)) \) the set of Pareto optimal points for \( s_{\psi,\lambda}(G, N) \). We present an algorithm \( A_5 \) that builds the set \( Par(s_{\psi,\lambda}(G, M)) \). More precisely, it constructs the set \( B(N) = Par(s_{\psi,\lambda}(G, N)) \) for each node \( N \) of \( G \).

**Algorithm \( A_5 \) (construction of POPs for element partition trees)**

**Input:** Increasing cost functions for element partition trees \( \psi \) and \( \lambda \) given by triples of functions \( \psi^0, F, w \) and \( \lambda^0, H, u \), respectively, a mesh \( M \), and the graph \( G = \Delta (M) \).

**Output:** The set \( Par(s_{\psi,\lambda}(G, M)) \) of Pareto optimal points for the set of pairs \( s_{\psi,\lambda}(G, M) = \left\{ (\psi(M, M, \Gamma), \lambda(M, M, \Gamma)) : \Gamma \in Etree(G, M) \right\} \).

1. If all nodes in \( G \) are processed, then return the set \( B(M) \). Otherwise, choose a node \( N \) in the graph \( G \) which is not processed yet and which is either a terminal node (unitary submesh of \( M \)) or an internal node (nonunitary submesh of \( M \)) such that, for any \( l \in DL(N) \) and any \( \delta \in \{0, 1\} \), the node \( N(l, \delta) \) is already processed, i.e. the set \( B(N(l, \delta)) \) is already constructed.

2. If \( N \) is a terminal node, then set \( B(N) = \{ (\psi^0(M, N), \lambda^0(M, N)) \} \). Mark the node \( N \) as processed and proceed to the step 1.

3. If \( N \) is an internal node then, for each \( l \in DL(N) \), compute

\[ D(N, l) = B(N(l, 0)) \langle FH \rangle B(N(l, 1)), \]
apply algorithm $A_4$ to $D(N,l)$ to obtain $C(N,l) = Par(D(N,l))$, and compute

$$B(N,l) = C(N,l) \langle ++ \rangle \{(w(M,N,l), u(M,N,l))\}$$

$$= \{(a + w(M,N,l), b + u(M,N,l)) : (a, b) \in C(N,l)\} .$$

4. Construct the set $A(N) = \bigcup_{l \in DL(N)} B(N,l)$ and apply the algorithm $A_4$ to $A(N)$ producing the set $B(N) = Par(A(N))$. Mark the node $N$ as processed and return to the step 1.

**Proposition 22.** Let $\psi$ and $\lambda$ be increasing cost functions for element partition trees defined by triples of functions $\psi^0, F, w$ and $\lambda^0, H, u$, respectively, $M$ be a mesh, and $G = \Delta(M)$. Then, for each node $N$ of the graph $G$, the algorithm $A_5$ constructs the set $B(N) = Par(s_{\psi,\lambda}(G, N))$.

**Proof.** We prove the considered statement by induction on nodes of $G$. Let $N$ be a terminal node of $G$. Then $Etree(G, N) = \{etree(\varphi(N))\}$,

$$s_{\psi,\lambda}(G, N) = Par(s_{\psi,\lambda}(G, N)) = \{((\psi^0(M,N), \lambda^0(M,N))\} ,$$

and $B(N) = Par(s_{\psi,\lambda}(G, N))$.

Let $N$ be an internal node of $G$ such that, for any $l \in DL(N)$ and any $\delta \in \{0, 1\}$, the considered statement holds for the node $N(l, \delta)$, i.e.,

$$B(N(l, \delta)) = Par(s_{\psi,\lambda}(G, N(l, \delta))) .$$

Let $l \in DL(N)$, $P(l, \delta) = s_{\psi,\lambda}(G, N(l, \delta))$ for $\delta = 0, 1$, and

$$P(l) = (P(l, 0) \langle FH \rangle P(l, 1)) \langle ++ \rangle \{(w(M,N,l), u(M,N,l))\} .$$
One can show that

\[ Par(P(l)) = Par(P(l,0) \langle FH \rangle P(l,1)) \langle++\rangle \{(w(M,N,l),u(M,N,l))\} . \]

By Lemma 20, \( Par(P(l,0) \langle FH \rangle P(l,1)) \subseteq Par(P(l,0)) \langle FH \rangle Par(P(l,1)) \). It is clear that \( Par(P(l,0)) \langle FH \rangle Par(P(l,1)) \subseteq P(l,0) \langle FH \rangle P(l,1) \). Using Lemma 18, we obtain

\[ Par(P(l,0) \langle FH \rangle P(l,1)) = Par(Par(P(l,0)) \langle FH \rangle Par(P(l,1))) . \]

According to the inductive hypothesis, \( B(N(l,\delta)) = Par(P(l,\delta)) \) for \( \delta = 0,1 \). Therefore

\[ Par(P(l)) = Par(B(N(l,0)) \langle FH \rangle B(N(l,1))) \langle++\rangle \{(w(M,N,l),u(M,N,l))\} \]

and \( Par(P(l)) = B(N,l) \).

It is clear that \( s_{\psi,\lambda}(G,N) = \bigcup_{l \in DL(N)} P(l) \). By Lemma 19,

\[ Par(s_{\psi,\lambda}(G,N)) = Par( \bigcup_{l \in DL(N)} P(l)) \subseteq \bigcup_{l \in DL(N)} Par(P(l)) . \]

Since \( \bigcup_{l \in DL(N)} Par(P(l)) \subseteq s_{\psi,\lambda}(G,N) \), we deduce by Lemma 18 that

\[ Par(s_{\psi,\lambda}(G,N)) = Par( \bigcup_{l \in DL(N)} Par(P(l))) = Par(A(N)) = B(N) . \]

\[ \square \]

In the proof of the following statement we use notation from the description of the algorithm \( A_5 \).

**Proposition 23.** Let \( \psi \) and \( \lambda \) be increasing cost functions for element partition trees
defined by triples of functions $\psi^0, F, w$ and $\lambda^0, H, u$, respectively, and functions $\psi^0, F, w, \lambda^0, H, u$ can be computed in polynomial time depending on the size $s(M)$ of an input mesh $M$. Let the function $ub(\psi, M)$ be bounded from above by a polynomial on $s(M)$. Then the time complexity of the algorithm $A_5$ is polynomial depending on $s(M)$.

Proof. We consider computations of the functions $\psi^0, F, w, \lambda^0, H, u$, comparisons and additions as elementary operations. Let $M$ be a mesh, $G = \Delta(M)$, and $s_{\psi,\lambda}(G, N)^{(1)} = \{a : (a, b) \in s_{\psi,\lambda}(G, N)\}$ for any node $N$ of $G$. It is clear that $s_{\psi,\lambda}(G, N)^{(1)} \subseteq \{0, 1, \ldots, ub(\psi, M)\}$. From Proposition 22 it follows that $B(N) = \text{Par}(s_{\psi,\lambda}(G, N))$. Using Lemma 16 we obtain that $|B(N)| \leq ub(\psi, M) + 1$.

To process a terminal node $N$ of $G$ (to construct the set $B(N)$), the algorithm $A_5$ computes values $\psi^0(M, N)$ and $\lambda^0(M, N)$, i.e., makes two elementary operations.

Let $N$ be an internal node of $G$ such that, for any $l \in DL(N)$ and any $\delta \in \{0, 1\}$, the set $B(N(l, \delta))$ is already constructed. We know that $|B(N(l, \delta))| \leq ub(\psi, M) + 1$ for each $l \in DL(N)$ and $\delta \in \{0, 1\}$. Let $l \in DL(N)$. The algorithm $A_5$ performs $O(ub(\psi, M)^2)$ computations of $F$ and $H$ to construct $D(N, l)$. The cardinality of the set $D(N, l)$ is at most $(ub(\psi, M)+1)^2$. To find the set $C(N, l)$, the algorithm $A_5$ makes $O(ub(\psi, M)^2 \log ub(\psi, M))$ comparisons (see Proposition 17). To construct the set $B(N, l)$, the algorithm $A_5$ computes the values $w(M, N, l)$ and $u(M, N, l)$, and makes $O(ub(\psi, M)^2)$ additions. It is clear that $|DL(N)| \leq s(M)$. Therefore, to construct the set $A(N)$, the algorithm $A_5$ makes $O(s(M)ub(\psi, M)^2 \log ub(\psi, M))$ elementary operations. It is clear that $|A(N)| \leq s(M)(ub(\psi, M) + 1)^2$. To construct the set $B(N)$ from the set $A(N)$, the algorithm $A_5$ makes $O(s(M)ub(\psi, M)^2(\log s(M) + \log ub(\psi, M)))$ comparisons (see Proposition 17). Therefore, to construct the set $B(N)$ from the sets $B(N(l, \delta))$, the algorithm $A_5$ makes

$$O(s(M)ub(\psi, M)^2(\log s(M) + \log ub(\psi, M)))$$
elementary operations.

The graph $G$ contains $|\text{SUB}(M)|$ nodes. By Lemma 1, $|\text{SUB}(M)| \leq s(M)^4$. Therefore, to construct the set $B(M) = \text{Par}(s_{\psi,\lambda}(G, M))$, the algorithm $A_5$ makes $O(s(M)^5 ub(\psi, M)^2 (\log s(M) + \log ub(\psi, M)))$ elementary operations. Since each elementary operation can be done in polynomial time depending on $s(M)$, and $ub(\psi, M)$ is bounded from above by a polynomial on $s(M)$, the time complexity of the algorithm $A_5$ is polynomial depending on $s(M)$.

\[\square\]

**Corollary 24.** Let parameters $p$ and $q$ be fixed, and

$$\psi, \lambda \in \{\text{time}^{1,p,q}_{\text{mode}}, \text{time}^{\infty,p,q}_{\text{mode}}, \text{memory}^p_{\text{mode}} : \text{mode} \in \{KB, ASAP\}\}.$$ 

Then the algorithm $A_5$ has polynomial time complexity depending on the size $s(M)$ of the input mesh $M$.

**Proof.** Let $\psi$ and $\lambda$ be defined by triples of functions $\psi^0, F, w$ and $\lambda^0, H, u$, respectively. From Proposition 3 it follows that the functions $\psi^0, F, w, \lambda^0, H,$ and $u$ can be computed in polynomial time depending on the size $s(M)$ of the input mesh $M$. In addition, Lemmas 5 and 6 show that the function $ub(\psi, M)$ is bounded from above by a polynomial depending on $s(M)$. Using Proposition 23 we obtain that the algorithm $A_5$ has polynomial time complexity depending on the size $s(M)$ of the input mesh $M$.

\[\square\]

Let $\psi$ and $\lambda$ be increasing cost functions for element partition trees, $M$ be a mesh, and $G = \Delta(M)$.

To study relationship between cost functions $\psi$ and $\lambda$ on the set of element partition trees $\text{Etree}(G, M)$ we consider partial function $\mathcal{E}_{\psi,\lambda}^M : \mathbb{R} \to \mathbb{R}$ defined as follows:

$$\mathcal{E}_{\psi,\lambda}^M(x) = \min \{ \lambda(M, M, \Gamma) : \Gamma \in \text{Etree}(G, M), \psi(M, M, \Gamma) \leq x \}.$$
Let \((a_1, b_1), \ldots, (a_k, b_k)\) be the normal representation of the set \(\text{Par}(s_{\psi,\lambda}(G, M))\)

where \(a_1 < \ldots < a_k\) and \(b_1 > \ldots > b_k\).

By Lemma 21, for any \(x \in \mathbb{R}\),

\[
\mathcal{E}_{\psi,\lambda}^M(x) = \begin{cases} 
\text{undefined}, & x < a_1 \\
 b_1, & a_1 \leq x < a_2 \\
 \vdots & \vdots \\
b_{k-1}, & a_{k-1} \leq x < a_k \\
b_k, & a_k \leq x 
\end{cases}
\]

### 4.3 Experimental Results

In this section, we describe some experiments that show the usage of our presented tools. We consider a pair of criteria reflecting the number of floating point operations done in parallel and the memory consumption respectively. We compute the Pareto front (the set of Pareto optimal points) of these two criteria in both \(KB\) and \(ASAP\) mode for the three classes of meshes studied in previous chapter: \(P_k, E_k\) and \(A_k\). We present in Tables 4.1-4.6 the number of points in each Pareto front. We show in Figs. 4.1-4.3 some samples of the Pareto front computed and parts of the graphs describing the relationships between the considered cost functions (we do not draw graphs after the last Pareto optimal point).

The number of points in Pareto front is equal to one if and only if there exists a totally optimal element partition tree relative to the considered cost functions. The obtained results for totally optimal element partition trees are consistent with the results obtained by multi-stage optimization (see Tables 3.7, 3.9, and 3.11).

The size of Pareto front for \(E_k\) and \(A_k\) is irregular (see Tables 4.2, 4.3, 4.5, and 4.6).

We implemented our bicriteria optimization algorithm in Java. A Mac Pro desktop
Table 4.1: Size of Pareto front: $\text{time}_{\text{ASAP}}^{\infty,p,10^7}$ vs $\text{memory}_{\text{ASAP}}^p$ for $P_k$

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 4.2: Size of Pareto front: $\text{time}_{\text{ASAP}}^{\infty,p,10^7}$ vs $\text{memory}_{\text{ASAP}}^p$ for $E_k$

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>14</td>
<td>14</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>24</td>
<td>27</td>
<td>28</td>
<td>26</td>
</tr>
<tr>
<td>7</td>
<td>24</td>
<td>23</td>
<td>24</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>23</td>
<td>28</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>17</td>
<td>17</td>
<td>21</td>
<td>26</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>15</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

with 16 GB of RAM memory and 2.5 GHz Intel Core i7 is used for conducting the experiments. The runtime of the program is acceptable as table 4.7 shows the runtime of the program to find the set of Pareto optimal points corresponding to costs element partition trees for $A_k$ meshes with respect to $\text{time}_{\text{ASAP}}^{\infty,p,10^7}$ and $\text{memory}_{\text{ASAP}}^p$. The runtime is measured in seconds and computed as average of ten executions of the program. We note that the size of the mesh $A_k$ grows exponentially depending on the number of refinement steps ($k$).
Table 4.3: Size of Pareto front: $\text{time}^{\infty,p,107}_{ASAP}$ vs $\text{memory}^p_{ASAP}$ for $A_k$

<table>
<thead>
<tr>
<th></th>
<th>$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>38</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 4.4: Size of Pareto front: $\text{time}^{\infty,p,107}_{KB}$ vs $\text{memory}^p_{KB}$ for $P_k$

<table>
<thead>
<tr>
<th></th>
<th>$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>
### Table 4.5: Size of Pareto front: $time_{KB}^{\infty,p,107}$ vs $memory_{KB}^p$ for $E_k$

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

### Table 4.6: Size of Pareto front: $time_{KB}^{\infty,p,107}$ vs $memory_{KB}^p$ for $A_k$

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.5: Size of Pareto front: $time_{KB}^{\infty,p,107}$ vs $memory_{KB}^p$ for $E_k$

Table 4.6: Size of Pareto front: $time_{KB}^{\infty,p,107}$ vs $memory_{KB}^p$ for $A_k$
Figure 4.1: Pareto front and relationship: $time_{ASAP}^{\infty,107}$ vs $memory_{ASAP}^A$ for $P_4$

Figure 4.2: Pareto front and relationship: $time_{ASAP}^{\infty,107}$ vs $memory_{ASAP}^A$ for $E_4$
Figure 4.3: Pareto front and relationship: $time_{ASAP}^{4,107}$ vs $memory_{ASAP}^4$ for $A_4$

<table>
<thead>
<tr>
<th>$k$</th>
<th>$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0005</td>
</tr>
<tr>
<td>2</td>
<td>0.0024</td>
</tr>
<tr>
<td>3</td>
<td>0.0053</td>
</tr>
<tr>
<td>4</td>
<td>0.0178</td>
</tr>
<tr>
<td>5</td>
<td>0.0797</td>
</tr>
<tr>
<td>6</td>
<td>0.3599</td>
</tr>
<tr>
<td>7</td>
<td>1.5795</td>
</tr>
<tr>
<td>8</td>
<td>5.7443</td>
</tr>
<tr>
<td>9</td>
<td>24.3678</td>
</tr>
<tr>
<td>10</td>
<td>134.9378</td>
</tr>
</tbody>
</table>

Table 4.7: Execution time of bi-criteria optimization for $A_k$ mesh with respect to $time_{ASAP}^{\infty,p,107}$ and $memory_{ASAP}^p$. 
Part II

Adaptive and Non-Adaptive Algorithms
Chapter 5

Models of Computations

There are many ways of studying the complexity of a given problem. One way is to consider an algorithm that solves this problem correctly and analyze its time complexity. This provides an upper bound on the complexity of the studied problem.

Proving lower bounds is a different method for tackling complexity analysis. The aim of such technique is to show that all algorithms that solve the problem correctly have at least a particular complexity threshold. In order to be able to have such statements, we need to specify a concrete model of computation that represents all algorithms for a given problem. There are many available models of computation such as Turing machines, circuits, decision tests, decision trees, etc.

This chapter lays out the foundations needed to present results on the complexity of two problems: sorting and majority presented in Chapter 6 in addition to an algorithm that is used to find optimal non-adaptive algorithms proposed in Chapter 7. It starts by introducing the notion of problem complexity in Section 5.1. Then a unified representation of different problems in the form of decision tables is discussed in Section 5.2. Section 5.3 defines decision trees for decision tables that can model adaptive algorithms. Finally, the chapter is concluded with a presentation of decision tests for decision tables in Section 5.4. These can be thought of as a non-adaptive model of computation.
5.1 Introduction

The worst-case time complexity of an algorithm $A$ on all inputs of size $n$ is defined as the maximum runtime of $A$ on any input of size $n$. This can be formally stated as follows:

$$T_A(n) = \max_{|X|=n} T_A(X).$$

We may be interested in a different case in which we need to determine the worst-case complexity of a given problem $\pi$. In other words, we need to make statements about how fast can we solve this problem. The worst-case complexity of $\pi$ (denoted by $T_\pi(n)$) for inputs of size $n$ is defined as the worst-case time complexity of the fastest algorithm $A$ that correctly solves this problem:

$$T_\pi(n) = \min_{A \text{ solves } \pi} T_A(n).$$

We define an upper bound on the complexity of a problem $\pi$ by presenting an algorithm $A$ that correctly solves this problem. The worst-case time complexity of $A$ forms an upper bound on the worst-case complexity of $\pi$:

$$T_\pi(n) \leq T_A(n).$$

To improve the upper bound, we need to design a faster algorithm that solves the problem. On the other hand, we may tackle the problem from another angle by considering lower bounds. We can study the problem itself and show that the problem is hard by proving that it can not be solved in less than some amount of steps $T(n)$:

$$T(n) \leq T_\pi(n).$$

This is a much harder task as we need to show that no algorithm among all
algorithms that solves this problem correctly can solve it in less than the given lower bound.

To introduce a statement about all algorithms, we need to have a representation of any such algorithm. Models of computation provide such representation by describing the class of algorithms considered in addition to a measure of their complexity. A more detailed discussion of lower bounds is presented in [47].

When we construct an algorithm $A$ whose worst-case runtime is the same as lower bounds on the considered problem $\pi$, then $A$ is an optimal algorithm for $\pi$. We consider this situation with the modified majority problem described in Section 6.2.

5.2 Problems as Decision Tables

This section describes the notion of decision tables and shows how different problems can be modeled as decision tables. We start with a simple general problem $z$ that is discussed in details in [48]. We may use this important problem to model many other different problems.

Let $U$ be a set of inputs that denote the universe. We define $f_1, \ldots, f_n$ as conditional attributes such that each attribute $f_i$ is a function that maps inputs from $U$ to a finite subset of the set of nonnegative integers $\omega = \{0, 1, \ldots\}$. If the range of the attribute $f_i$ is a set $V_i = \{v_1, \ldots, v_k\}$ then $f_i$ divides the domain into $k$ subsets where the value of this attribute in the $i$th subset is $v_i$. All attributes $f_1, \ldots, f_n$ divide the universe $U$ into different domains where the values of the attributes in each of these domains are constant. Each domain is assigned a label from $\omega$ and different domains may have the same label.

A special case of this problem occurs when the possible value of any attribute $f_i$ belongs to $\{0, 1\}$. In this case, each attribute $f_i$ divides the domain into two parts: one in which the value of this attribute is zero and the other in which its value is one.

The goal of the problem introduced is to identify the label of the domain of an
element \( u \in U \) by asking about the values of the attributes on \( u \). We define the problem formally as a tuple \( z = (v, f_1, \ldots, f_n, U) \) where \( v \) is a function that maps \( V_1 \times \ldots \times V_n \) to \( \omega \). Given an element \( u \in U \), we need to find the label of its domain \( z(u) \) such that:

\[
z(u) = v(f_1(u), \ldots, f_n(u)).
\]

A decision table \( T \) is a rectangular table with \( n \geq 1 \) columns labeled with conditional attributes \( f_1, \ldots, f_n \). The table is filled with numbers from \( \omega \) and rows of this table can be interpreted as tuples of values of conditional attributes. Rows of the table are pairwise different and each row is labeled with a number from \( \omega \) that is referred to as the row decision. The column representing the decision values of the rows is referred to as the decision attribute \( d(T) \) and we denote the set of values of the decision attribute by \( D(T) \).

A decision table is empty if it has no rows. We define a degenerate decision table as one that is empty or whose all rows have the same decision. We denote by \( N(T) \) the number of rows of \( T \), \( \dim(T) \) as the number of conditional attributes of \( T \). For a conditional attribute \( f_i \) of \( T \), the set of values of \( f_i \) associated with rows of \( T \) is referred to as \( E(T, f_i) \). \( E(T) \) is the set of conditional attributes \( f_i \) of \( T \) whose range of values has size at least two (\( |E(T, f_i)| \geq 2 \)). We define a separable subtable \( T' = T(f_{i_1}, \delta_1) \ldots (f_{i_k}, \delta_k) \) of \( T \) as the subtable of \( T \) containing only rows of \( T \) whose values of conditional attributes \( f_{i_1}, \ldots, f_{i_k} \) are \( \delta_1, \ldots, \delta_k \), respectively.

We use a decision table \( T_z \) as a way of representation of the general problem \( z \) introduced above. We consider rows as objects of the universe that we know their labels. Columns of the table represent the conditional attributes associated with the problem and the decision column corresponds to the labels of objects of the universe.

More formally, given a problem \( z = (v, f_1, \ldots, f_n, U) \) we may use a decision table \( T_z \) to represent this problem. Columns of \( T_z \) correspond to the conditional attributes
\( f_1, \ldots, f_n \) of problem \( z \). For any element \( u \in U \) of the problem’s universe with label \( z(u) \), we associate a row \( \delta = (f_1(u), \ldots, f_n(u)) \) with decision \( v(f_1(u), \ldots, f_n(u)) \). We replace group of equal rows by one row labeled with their common decision.

We use decision trees and tests for a decision table as possible algorithms for solving the problem represented by this table as explained next.

### 5.3 Decision Trees

A decision tree \( \Gamma \) over a decision table \( T \) is a finite directed rooted tree. Leaves of \( \Gamma \) (we call them also terminal nodes) are labeled with a number from the set of nonnegative integers \( \omega \). Its internal nodes are labeled with conditional attributes of \( T \). Edges start in each internal node labeled with pairwise different nonnegative integers.

We associate with each node \( \eta \) of \( \Gamma \) a separable subtable \( T(\eta) \) of \( T \) as follows. If \( \eta \) is the root of \( \Gamma \), then \( T(\eta) = T \). Otherwise, let \( \eta_1, e_1, \ldots, \eta_k, e_k, \eta \) be the directed path from the root of \( \Gamma \) to \( \eta \), then \( T(\eta) = T(f_{i_1}, \delta_1) \ldots (f_{i_k}, \delta_k) \). Note that \( f_{i_1}, \ldots, f_{i_k} \) denote attributes corresponding to labels of the nodes \( \eta_1, \ldots, \eta_k \) in the path and \( \delta_1, \ldots, \delta_k \) denote the values of these attributes described by the label of the edges \( e_1, \ldots, e_k \) along this path.

Given a decision table \( T \) and a decision tree \( \Gamma \) over \( T \), \( \Gamma \) is a decision tree for \( T \) under the following conditions. For any terminal node \( \eta \) of \( \Gamma \), \( T(\eta) \) is a nonempty degenerate separable subtable of \( T \) and the label of the node \( \eta \) is the value of the decision of all rows of \( T(\eta) \). For any internal node \( \eta \) of \( \Gamma \), the label of \( \eta \) is an attribute \( f_i \in E(T(\eta)) \) and for each value \( x \in E(T(\eta), f_i) \) we have an edge coming out of \( \eta \) labeled with \( x \). Let \( r \) be a row of \( T \) and \( \eta \) be a terminal node of \( \Gamma \). We say that the path from the root of \( \Gamma \) to \( \eta \) corresponds to the work of \( \Gamma \) on the row \( r \) (accepts the row \( r \)) if \( r \) belongs to \( T(\eta) \).

Decision trees describe adaptive algorithms in a natural way. This is due to the
fact that the $i$th question of the algorithm on a given input varies according to the answers received in the previous $i - 1$ questions. A decision tree $\Gamma'$ that describes adaptive algorithms for a problem $z$ have all of its terminal nodes labeled with labels of different domains of the universe of $z$. Internal nodes are labeled with conditional attributes of $z$ and edges coming out of these nodes are pairwise different and labeled with possible values of this conditional attribute.

We define the work of $\Gamma'$ on an input $u \in U$ of $z$ as follows. The work starts at the root of the tree. If the considered node is a terminal node then the result of work of $\Gamma'$ on $u$ is the label of this node. Otherwise the considered node is labeled with a conditional attribute $f_i$ of $z$ and the work of $\Gamma'$ follows by working on the subtree connected by the edge labeled with $f_i(u)$.

A decision tree $\Gamma'$ for $z$ represents a correct adaptive algorithm for $z$ iff the work of $\Gamma'$ on any input $u \in U$ is $v(f_1(u), \ldots, f_n(u))$. We model adaptive algorithms that solve problem $z$ by decision trees for the equivalent decision table $T_z$ representing the problem $z$. It is clear that there is a one-to-one correspondence between correct adaptive algorithms for problem $z$ and decision trees for $T_z$.

We define the depth of the decision tree $\Gamma$, $h(\Gamma)$, to be the the maximum length of a path from the root to a terminal node of $\Gamma$. We denote by $h(T_z)$ the minimum depth of a decision tree $\Gamma$ that solves the problem $z$ represented by the decision table $T_z$. This complexity measure reflects the worst-case time complexity of the problem $z$ as it indicates the minimum number of questions required to determine the label of any element $u \in U$ of $z$.

Another complexity measure of $\Gamma$ is its average depth $h_{\text{avg}}(\Gamma)$. We define the total path length of $\Gamma$ as the sum of the lengths of the paths corresponding to work of $\Gamma$ on each row of $T$. We consider the average depth as the total path length divided by the number of rows. We assume that all rows of the table $T$ are equally likely to appear. We denote by $h_{\text{avg}}(T_z)$ the minimum average depth of a decision tree $\Gamma$ that
solves the problem \( z \) represented by \( T_z \). Assuming that elements of different domains of the universe are equally likely to appear, this complexity measure highlights the average time complexity of an algorithm solving the problem \( z \).

### 5.3.1 Optimization of Decision Trees

Many problems involving optimization of decision trees are NP-hard [49, 50]. There are well-known exact algorithms for optimizing decision trees. Those algorithms are based on various techniques: brute-force [51], dynamic programming [14, 52, 53], and branch-and-bound [54].

During our work with decision trees, we use an approach for the optimization of decision trees based on extensions of dynamic programming. This approach [13, 14, 55] relies on multi-stage optimization of decision trees based on different criteria. It considers separable subtables of the initial decision table as individual subproblems. Since the number of separable subtables of a decision table may be very large, this approach can be applied directly to relatively small decision tables.

### 5.4 Decision Tests

Given a decision table \( T \) with \( n \) conditional attributes \( f_1, \ldots, f_n \), a decision test (test) \( \{f_{i_1}, \ldots, f_{i_m}\} \) for \( T \) is a subset of conditional attributes that are sufficient to separate any pair of rows of \( T \) with different decisions. A conditional attribute \( f_i \) of \( T \) separates a pair of rows if they have different values with respect to this conditional attribute. A subset of conditional attributes separates a pair of rows of \( T \) if at least one of its attributes separates this pair.

We define \( R(T) \) to be the minimum cardinality of a test of \( T \). A test whose each proper subset is not a test is called a reduct. It is clear that a test with minimum cardinality is a reduct.

A non-adaptive (oblivious) algorithm specifies all of its questions before it is given
any input. In other words, the $i$th question of such algorithm does not depend on the answers received for its previous $i - 1$ questions.

Decision test models non-adaptive algorithms for a given problem

$$z = (v, f_1, \ldots, f_n, U)$$

in an intuitive way. A decision test for a problem $z$ is defined as a subset $\{f_{i_1}, \ldots, f_{i_m}\}$ of the conditional attributes of $z$. A decision test for a problem $z$ corresponds to a correct non-adaptive algorithm of $z$ if it is sufficient to correctly recognize the label $z(u)$ of any element $u \in U$. Formally, $\{f_{i_1}, \ldots, f_{i_m}\}$ defines a correct non-adaptive algorithm for $z$ if there is a mapping $\mu : V_{i_1} \times \ldots \times V_{i_m} \rightarrow \omega$ such that for any $u \in U$:

$$\mu(f_{i_1}(u), \ldots, f_{i_m}(u)) = v(f_1(u), \ldots, f_n(u)).$$

It is clear that there is a one-to-one correspondence between decision tests for $T_z$ and correct non-adaptive algorithms for $z$.

Finally, we can represent any test $\{f_{i_1}, \ldots, f_{i_m}\}$ as a special decision tree $\Gamma$ whose all nodes in level $j = 1, \ldots, m$ are labeled with the attribute $f_{i_j}$. Terminal nodes of $\Gamma$ are in level $m + 1$ and they are labeled as follows. Let us consider a path from the root of $\Gamma$ to a terminal node $l$ in which edges are labeled with numbers $\delta_1, \ldots, \delta_m$. We assign to the terminal node $l$ the label $\mu(\delta_1, \ldots, \delta_m)$.

### 5.4.1 Optimization of Decision Tests

The problem of finding minimum decision tests is NP-hard [19, 48]. We suggest an algorithm for finding minimum tests for decision tables in Chapter 7.
Chapter 6

Fundamental Combinatorial Problems

This chapter starts with a description of results for the eight element sorting problem under the decision tree model of computation in Section 6.1. Those results are obtained using the techniques of multi-stage optimization of decision trees discussed previously in Section 5.3.1. In addition, hypotheses about complexity of a modified majority problem are formulated with the help of computer experiments and proved analytically in Section 6.2. Classes of algorithms studied are adaptive algorithms modeled as decision trees and non-adaptive algorithms modeled as decision tests.

One common step of our experimental work with the sorting and modified majority problems is to represent the problem in the form of a decision table as described in Section 5.2. Attributes of the table denote possible steps of algorithms solving the problem. For example, such steps may be comparing two elements when studying comparison-based algorithms for sorting. Rows represent possible inputs for the problem and they are labeled with the corresponding output. We describe the corresponding decision table in an explicit way for the modified majority problem. For the sorting problem, we use a decision table representation in the computer experiments with DAGGER but we do not describe it here.

6.1 Sorting

We prove that the minimum average depth of a decision tree for sorting 8 pairwise different elements is equal to $\frac{620160}{8!}$. We show also that each decision tree for
sorting 8 elements, which has minimum average depth (the number of such trees is approximately equal to $8.548 \times 10^{326365}$), has also minimum depth. Both problems were considered by Knuth in [11].

### 6.1.1 Introduction

The problem of sorting $n$ pairwise different elements from a linearly ordered set is one of the model problems in algorithm theory [11]. For solving this problem, we use binary decision trees [50, 56] where each step is a comparison of two elements. The minimum number of nodes in such tree is equal to $2(n!) - 1$. For $n = 2, \ldots, 11$, the minimum depth of a decision tree for sorting $n$ elements is equal to the well known lower bound $\lceil \log_2(n!) \rceil$ [57]. For $n = 2, 3, 4, 5, 6, 9, 10$, the minimum average depth of a decision tree for sorting $n$ elements is equal to the known lower bound $\varphi(n)/n!$, where $\varphi(n) = (\lceil \log_2 n! \rceil + 1) \cdot n! - 2^{\lceil \log_2 n! \rceil}$ is the minimum external path length in an extended binary tree with $n!$ terminal nodes [11]. Césary [58] proved that, for $n = 7$ and $n = 8$, there are no decision trees for sorting $n$ elements whose average depth is equal to $\varphi(n)/n!$. Kollár [59] found that the minimum average depth of a decision tree for sorting 7 elements is equal to $62416/7!$. We find that the minimum average depth of a decision tree for sorting 8 elements is equal to $620160/8!$.

Another open problem considered by Knuth [11] is the existence of decision trees for sorting $n$ elements which have simultaneously minimum average depth and minimum depth. As it was mentioned by Knuth in [11], if a decision tree for sorting $n$ elements has average depth equal to $\varphi(n)/n!$ then this tree has depth equal to $\lceil \log_2 n! \rceil$. Therefore, for $n = 2, 3, 4, 5, 6, 9, 10$, each decision tree for sorting $n$ elements, which has minimum average depth, has also minimum depth. We extended this result to the cases $n = 7$ (Kollár in [59] did not consider this question) and $n = 8$. For $n = 2, \ldots, 8$, we counted also the number of decision trees for sorting $n$ elements which have minimum average depth. In particular, for $n = 8$, the number of such
trees is approximately equal to $8.548 \times 10^{326365}$. We recalculate known values of the minimum depth for $n = 2, \ldots, 8$ and minimum average depth for $n = 2, \ldots, 7$ to make sure that the new results are valid.

To obtain these results, we use tools from the system DAGGER which is based on extensions of dynamic programming \cite{13, 14, 55} that allow us to make sequential optimization of decision trees relative to depth and average depth, and to count the number of decision trees with minimum average depth. The considered algorithms are not, of course, brute-force algorithms (it is impossible to work directly with $8.548 \times 10^{326365}$ optimal trees for $n = 8$). However, they require the work with big number of subproblems (for $n = 8$, the number of subproblems is equal to 431,723,379). We describe the notion of subproblems in Section 6.1.2. The construction of the directed acyclic graph (DAG) representing the ordering of subproblems is discussed in Section 6.1.3.

The time complexity of the functions provided by the system DAGGER depend on the number of nodes and edges of the DAG corresponding to the input problem.

Section 6.1.2 contains main results and Section 6.1.3 – description of tools.

### 6.1.2 Main Results

Let $x_1, \ldots, x_n$ be pairwise different elements from a linearly ordered set. We should find a permutation $(p_1, \ldots, p_n)$ from the set $P_n$ of all permutations of the set $\{1, \ldots, n\}$ such that $x_{p_1} < \ldots < x_{p_n}$. Each nonempty subset $Q$ of the set $P_n$ can be considered as a subproblem of the initial sorting problem $P_n$ with inputs $x_1, \ldots, x_n$ for each of which there exists a permutation $(p_1, \ldots, p_n) \in Q$ such that $x_{p_1} < \ldots < x_{p_n}$. We give all required definitions not for $P_n$ but for an arbitrary subset (subproblem) $Q$ of $P_n$.

We denote by $I(n)$ the set of all inequalities of the form $x_i < x_j$ such that $(i, j) \in \pi(n) = \{(i, j) : 1 \leq i, j \leq n, i \neq j\}$. We say that the permutation $p = (p_1, \ldots, p_n)$ is compatible with the inequality $x_i < x_j$ if and only if $i$ precedes $j$ in $p$. For
For solving the subproblem $Q$, we use binary decision trees in which terminal nodes are labeled with permutations from $Q$. Each nonterminal node is labeled with a comparison $x_i : x_j$ of two elements where $(i, j) \in \pi(n)$. Two edges start in this node which are labeled with results of the comparison $x_i < x_j$ and $x_j < x_i$, respectively.

We denote by $E(Q)$ the set of comparisons $x_i : x_j$ such that $(i, j) \in \pi(n)$, $Q(x_i < x_j) \neq \emptyset$ and $Q(x_j < x_i) \neq \emptyset$.

Let $\Gamma$ be a decision tree and $v$ be a node of $\Gamma$. We denote $Q(v) = Q(s_1) \ldots (s_m)$ where $s_1, \ldots, s_m$ are all inequalities attached to the edges in the path from the root of $\Gamma$ to $v$ (if $v$ is the root of $\Gamma$ then $Q(v) = Q$).

We will say that $\Gamma$ solves the subproblem $Q$ if each node $v$ of $\Gamma$ satisfies the following conditions:

1. If $|Q(v)| = 1$ and $Q(v) = \{p\}$ then $v$ is a terminal node labeled with the permutation $p$;

2. If $|Q(v)| > 1$ then the node $v$ is a nonterminal node which is labeled with a comparison $x_i : x_j$ from the set $E(Q(v))$.

We consider three cost functions for decision trees. Let $\Gamma$ be a decision tree for solving the subproblem $Q$. We denote by $h(\Gamma)$ the depth of $\Gamma$ which is the maximum length of a path from the root to a terminal node, by $l(\Gamma)$ – the external path length in $\Gamma$ (the sum of lengths of all paths from the root to terminal nodes of $\Gamma$), and by $h_{avg}(\Gamma)$ – the average depth of $\Gamma$ which is equal to $l(\Gamma)/|Q|$ (one can show that each decision tree for solving $Q$ has $|Q|$ terminal nodes).

We denote by $h(Q)$ the minimum depth, by $l(Q)$ – the minimum external path length, and by $h_{avg}(Q)$ – the minimum average depth of decision trees for solving the subproblem $Q$. Note that $h_{avg}(Q) = l(Q)/|Q|$. 
For \( n = 2, \ldots, 8 \), the values \( h(P_n) \) and \( h_{\text{avg}}(P_n) = l(P_n)/n! \) can be found in Tables 6.1 and 6.2. For the considered values of \( n \), the parameter \( h(P_n) \) is equal to its lower bound \( \lceil \log_2(n!) \rceil \), and the parameter \( h_{\text{avg}}(P_n) \) is equal to its lower bound \( \varphi(n)/n! \) for \( n = 2, \ldots, 6 \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h(P_n) )</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>( h_{\text{avg}}(P_n) )</td>
<td>2/2!</td>
<td>16/3!</td>
<td>112/4!</td>
<td>832/5!</td>
</tr>
<tr>
<td>( \varphi(n)/n! )</td>
<td>2/2!</td>
<td>16/3!</td>
<td>112/4!</td>
<td>832/5!</td>
</tr>
<tr>
<td>(</td>
<td>Opt_{h_{\text{avg}}}(P_n)</td>
<td>)</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>(</td>
<td>\mathcal{P}(n)</td>
<td>)</td>
<td>3</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 6.1: Results for sorting \( n = 2, 3, 4, 5 \) elements.

Let \( \Gamma \) be a decision tree for solving the subproblem \( Q \) and \( \psi \) be one of the cost functions \( h, l, h_{\text{avg}} \). We will say that \( \Gamma \) is optimal relative to \( \psi \) if \( \psi(\Gamma) = \psi(Q) \). We denote by \( Opt_{\psi}(Q) \) the set of decision trees for the subproblem \( Q \) which are optimal relative to \( \psi \).

It is clear that \( Opt_{h_{\text{avg}}}(P_n) = Opt_{l}(P_n) \). Based on results of computer experiments we obtain that \( Opt_{h_{\text{avg}}}(P_n) \subseteq Opt_{h}(P_n) \) for \( n = 2, \ldots, 8 \). For \( n = 2, \ldots, 8 \), we count also the cardinality of the set \( Opt_{h_{\text{avg}}}(P_n) \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h(P_n) )</td>
<td>10</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>( h_{\text{avg}}(P_n) )</td>
<td>6896/6!</td>
<td>62416/7!</td>
<td>620160/8!</td>
</tr>
<tr>
<td>( \varphi(n)/n! )</td>
<td>6896/6!</td>
<td>62368/7!</td>
<td>619904/8!</td>
</tr>
<tr>
<td>(</td>
<td>Opt_{h_{\text{avg}}}(P_n)</td>
<td>)</td>
<td>( 1.968 \times 10^{263} )</td>
</tr>
<tr>
<td>(</td>
<td>\mathcal{P}(n)</td>
<td>)</td>
<td>130,023</td>
</tr>
</tbody>
</table>

Table 6.2: Results for sorting \( n = 6, 7, 8 \) elements.

A nonempty subproblem \( Q \subseteq P_n \) is called a separable subproblem of \( P_n \) if there exists a subset \( I \) of the set of inequalities \( I(n) \) such that \( Q \) is the set of all permutations from \( P_n \) which are compatible with each inequality from \( I \). In particular, \( Q = P_n \) if \( I = \emptyset \). We denote by \( \mathcal{P}(n) \) the set of all separable subproblems of \( P_n \). The cardinality of the set \( \mathcal{P}(n) \) for \( n = 1, \ldots, 8 \) can be found in Tables 6.1 and 6.2.
All computations were done using our software system DAGGER [12] for optimization of decision trees and rules. This system is based on extensions of dynamic programming that allow us to describe the set of all decision trees for the initial problem, to make sequential optimization relative to different cost functions and to count the number of optimal decision trees for some cost functions. The work of DAGGER involves the construction and transformations of a directed acyclic graph whose nodes are subproblems of the initial problem. In the case of sorting \( n \) elements, the set of nodes coincides with the set \( \mathcal{P}(n) \) of separable subproblems of \( P_n \).

6.1.3 Tools

Graph \( \Delta_n \) and its Proper Subgraphs

We now consider an algorithm for the construction of a directed acyclic graph \( \Delta_n \), which can represent the set of all decision trees for sorting \( n \) elements. Nodes of this graph are separable subproblems of \( P_n \). During each step we process one node and mark it with symbol *. We start with the graph that consists of one node \( P_n \) and finish when all nodes of the graph are processed.

Let the algorithm have already performed \( t \) steps. We now describe the step number \((t + 1)\). If all nodes are processed then the algorithm terminates, and the resulted graph is \( \Delta_n \). Otherwise, choose a node (subproblem) \( Q \) that has not been processed yet.

If \( |Q| = 1 \), mark the considered node with symbol * and proceed to the step number \((t + 2)\). Let \( |Q| > 1 \). For each \( x_i : x_j \in E(Q) \), draw a pair of edges from the node \( Q \) (this pair of edges will be called \( (x_i : x_j) \)-pair) and label these edges with the inequalities \( x_i < x_j \) and \( x_j < x_i \). These edges enter the nodes \( Q(x_i < x_j) \) and \( Q(x_j < x_i) \), respectively. If any of the nodes \( Q(x_i < x_j), Q(x_j < x_i) \) are not present in the graph then add these nodes to the graph. Mark the node \( Q \) with symbol * and proceed to the step number \((t + 2)\).
It is clear that $\Delta_n$ is a directed acyclic graph. A node of this graph is called terminal if it has no outgoing edges. A node $Q$ is terminal if and only if $|Q| = 1$. We define the size of $\Delta_n$ to be the number of its nodes and edges. The complexity of a given problem mainly depends on the size of its corresponding DAG.

We now introduce the notion of proper subgraph of the graph $\Delta_n$. For each node of the graph $\Delta_n$, which is not terminal, we can remove any but not all pairs of edges that leave the node. The obtained subgraph will be called a proper subgraph of the graph $\Delta_n$. It is clear that all terminal nodes of this subgraph are terminal nodes of the graph $\Delta_n$. We consider $\Delta_n$ as a proper subgraph of the graph $\Delta_n$. Proper subgraphs of the graph $\Delta_n$ can be obtained as results of procedures of decision tree optimization relative to the depth or average depth (external path length).

Let $G$ be a proper subgraph of the graph $\Delta_n$. Now, for each node $Q$ of the graph $G$, we describe the set of decision trees corresponding to it. We will move from terminal nodes to the node $P_n$. Let $Q$ be a terminal node and $Q = \{p\}$. Then the only trivial decision tree depicted in Figure 6.1 corresponds to the considered node.

Let $Q$ be a nonterminal node. Then there is a number of pairs of edges starting in $Q$. We consider an arbitrary pair and describe the set of decision trees corresponding to this pair. Let the considered pair be an $(x_i : x_j)$-pair where $x_i : x_j \in E(Q)$, and $\Gamma_1$ and $\Gamma_2$ be decision trees from the sets corresponding to the nodes $Q(x_i < x_j)$ and $Q(x_j < x_i)$, respectively. Then the decision tree depicted in Figure 6.2 belongs to the set of decision trees, which correspond to this pair. All such decision trees belong to
the considered set, and this set does not contain any other decision trees. Then the
set of decision trees corresponding to the node $Q$ coincides with the union of sets of
decision trees corresponding to pairs starting in $Q$. We denote by $D_G(Q)$ the set of
decision trees corresponding to the node $Q$ in the graph $G$.

Using the graph $G$ it is easy to find the number of decision trees in the set $D_G(Q)$:
$|D_G(Q)| = 1$ if $Q$ is a terminal node. Let us consider a node $Q$, which is not terminal,
and an $(x_i : x_j)$-pair of edges, which start in this node and enter the nodes $Q_1 = \n(x_i < x_j)$, $Q_2 = Q(x_j < x_i)$. We assign to this pair the number $|D_G(Q_1)| \times |D_G(Q_2)|$.
Then $|D_G(Q)|$ is equal to the sum of numbers corresponding to pairs starting in $Q$.

The next statement follows from the results obtained in [13].

**Proposition 25.** Let $Q$ be a node in the graph $\Delta_n$ (a separable subproblem of $P_n$).
Then the set $D_{\Delta_n}(Q)$ coincides with the set of all decision trees for solving the sub-
problem $Q$. In particular, the set $D_{\Delta_n}(P_n)$ coincides with the set of all decision trees
for sorting $n$ elements.

**Procedures of Optimization**

Let $G$ be a proper subgraph of the graph $\Delta_n$, and $\psi$ be either depth $h$ or external
path length $l$ (it is more convenient for us to work with $l$ than with $h_{avg}$). Below
we describe a procedure of decision tree optimization relative to $\psi$, which transforms
the graph $G$ into a proper subgraph $G^\psi$ of $G$. We begin from terminal nodes and
move to the node $P_n$. We attach a number $\psi(Q,G)$ to each node $Q$ which is equal
to the minimum value of $\psi$ for a decision tree from $D_G(Q)$, and we may remove
some pairs of edges, which start in the considered node. Let $Q$ be a terminal node.

We attach the number 0 to the node $Q$. Let us consider a node $Q$, which is not
terminal, and an $(x_i : x_j)$-pair of edges, which start in this node. Then the edges are
labeled with inequalities $x_i < x_j$, $x_j < x_i$, and enter nodes $Q(x_i < x_j)$, $Q(x_j < x_i)$,
respectively. Let the numbers $a_1, a_2$ be attached already to these nodes. Then we
attach to the considered pair of edges the number \(1 + \max\{a_1, a_2\}\) if \(\psi = h\), and the number \(|Q| + a_1 + a_2\) if \(\psi = l\).

Among numbers attached to pairs starting in \(Q\) we choose the minimum number \(a\) and attach it to the node \(Q\) as the number \(\psi(Q, G)\). We remove all pairs starting in \(Q\) to which numbers are attached that are greater than \(a\). When all nodes are treated we obtain a proper subgraph \(G^\psi\) of the graph \(G\).

The next statement follows from the results obtained in [13].

**Proposition 26.** Let \(\psi \in \{h, l\}\), \(G\) be a proper subgraph of the graph \(\Delta_n\) and \(Q\) be a node of \(G\). Then \(\psi(Q, G) = \min\{\psi(\Gamma) : \Gamma \in D_G(Q)\}\), \(D_{G^\psi}(Q) = \{\Gamma : \Gamma \in D_G(Q), \psi(\Gamma) = \psi(Q, G)\}\) if \(\psi = l\), and \(D_{G^\psi}(Q) \subseteq \{\Gamma : \Gamma \in D_G(Q), \psi(\Gamma) = \psi(Q, G)\}\) if \(\psi = h\).

**Tool Use**

For \(n = 2, \ldots, 8\), we construct the graph \(\Delta_n\) and find the number of its nodes which is equal to \(|P(n)|\). Then we apply to the graph \(\Delta_n\) the procedures of decision tree optimization relative to \(l\) and \(h\) separately. We obtain proper subgraphs \(\Delta^l_n\) and \(\Delta^h_n\) of the graph \(\Delta_n\) in which (by Propositions 25 and 26) nodes \(P_n\) are labeled with numbers \(l(P_n, \Delta_n) = l(P_n)\) and \(h(P_n, \Delta_n) = h(P_n)\), respectively. As a result, we obtain also the value \(h_{avg}(P_n) = l(P_n)/n!\). We count the cardinality of the set \(D_{\Delta^l_n}(P_n)\) of decision trees corresponding to the node \(P_n\) in the graph \(\Delta^l_n\). By Propositions 25 and 26,

\[|D_{\Delta^l_n}(P_n)| = |Opt_{h_{avg}}(P_n)|.\]

After that, we apply to the graph \(\Delta^l_n\) the procedure of decision tree optimization relative to \(h\). As a result, we obtain proper subgraph \((\Delta^l_n)^h\) of the graph \(\Delta^l_n\) with the number \(h(P_n, \Delta^l_n)\) attached to the node \(P_n\). We find that \(h(P_n, \Delta^l_n) = h(P_n)\) and \((\Delta^l_n)^h = \Delta^l_n\). Using Propositions 25 and 26, we obtain that all decision trees for sorting \(n\) elements with minimum average depth have the same depth equal to \(h(P_n)\). Therefore \(Opt_{h_{avg}}(P_n) \subseteq Opt_h(P_n)\).
For $n = 2, \ldots, 7$, all computations were done on a Mac Pro desktop with two 2.40GHz 6-core Intel Xeon processors and 64GB of RAM. For $n = 8$, we used Amazon cr1.8xlarge instance with two 2.60GHz 8-core Intel Xeon processors and 244GB of RAM (see http://aws.amazon.com/ec2/instance-types/ for details).

For $n = 2, \ldots, 7$, all computations were done directly as it was described above. For $n = 8$, we used a notion of equivalent subproblems and studied only nonequivalent separable subproblems defined by one inequality (the number of such subproblems is equal to one) and defined by two inequalities (the number of such subproblems is equal to three).

For a subproblem $Q$ of the problem $P_n$, we denote by $T(Q)$ the set of decision trees for the subproblem $Q$ solving. We say that two subproblems $Q_1$ and $Q_2$ of the problem $P_n$ are equivalent if $|Q_1| = |Q_2|$ and there exists a bijection (a one-to-one correspondence) $\varphi : T(Q_1) \to T(Q_2)$ which preserves depth and external path length of decision trees.

First, we show that, for any $i, j \in \{1, \ldots, n\}$, $i \neq j$, the subproblem $P_n(x_i < x_j)$ is equivalent to the subproblem $P_n(x_1 < x_2)$. Evidently, $|P_n(x_1 < x_2)| = |P_n(x_i < x_j)|$. As a bijection

$$\varphi : T(P_n(x_1 < x_2)) \to T(P_n(x_i < x_j))$$

we can use a mapping which transforms a decision tree for the subproblem $P_n(x_1 < x_2)$ solving into a decision tree for the subproblem $P_n(x_i < x_j)$ solving by changing indexes in comparisons, inequalities and permutations attached to nonterminal nodes, edges and terminal nodes as follows: $1 \to i$, $2 \to j$, $i \to 1$, and $j \to 2$.

Next, we show in a similar way that, for any $i, j, k, l \in \{1, \ldots, n\}$, $i \neq j$, $k \neq l$, $\{i, j\} \neq \{k, l\}$, the subproblem $P_n(x_i < x_j)(x_k < x_l)$ is equivalent to one of the four subproblems: $S^a_1 = P_n(x_1 < x_2)(x_3 < x_4)$, $S^a_2 = P_n(x_1 < x_2)(x_2 < x_3)$, $S^a_3 = P_n(x_1 < x_2)(x_1 < x_3)$, and $S^a_4 = P_n(x_2 < x_1)(x_3 < x_1)$.

After that, we show that subproblems $S^a_3$ and $S^a_4$ are equivalent. It is clear that
$S^n_3 = \{(p_n, \ldots, p_1) : (p_1, \ldots, p_n) \in S^n_1\}$. As a bijection $\varphi : T(S^n_3) \to T(S^n_4)$ we can use a mapping which transforms a decision tree $\Gamma$ for solving the subproblem $S^n_3$ into a decision tree $\varphi(\Gamma)$ for solving the subproblem $S^n_4$ in the following way. If an edge in $\Gamma$ is labeled with an inequality $x_i < x_j$ then the corresponding edge in $\varphi(\Gamma)$ is labeled with the inequality $x_j < x_i$. If a terminal node in $\Gamma$ is labeled with a permutation $(p_1, \ldots, p_n)$ then the corresponding terminal node in $\varphi(\Gamma)$ is labeled with the permutation $(p_n, \ldots, p_1)$.

We studied subproblems $S^n_1$, $S^n_2$, and $S^n_3$ in a way similar to the described above for $P_n$, and combined the obtained results. The numbers of separable subproblems for $S^n_1$, $S^n_2$, and $S^n_3$ are equal to 29668833, 12650470, and 50444492, respectively. The time of computation is 1848 seconds for $S^n_1$, 326 seconds for $S^n_2$, and 3516 seconds for $S^n_3$.

The study of separable subproblems of $P_n$ described by at most two equations allows us to understand the structure of the first two levels of decision trees solving $P_n$ and having minimum average depth. The root (the only node in the first level) can be labeled with an arbitrary comparison $x_i : x_j$ such that $i, j \in \{1, \ldots, n\}$, $i \neq j$. Any node in the second level can be labeled with an arbitrary comparison $x_k : x_l$ such that $k, l \in \{1, \ldots, n\}$, $k \neq l$ and $\{k, l\} \cap \{i, j\} = \emptyset$.

6.2 Modified Majority Problem

In this section, we study the complexity of adaptive and non-adaptive algorithms for a modified majority problem.

6.2.1 Introduction

Given an $n$-tuple of elements colored by one of two colors, we should find an element of the majority color if it exists or report that there is no majority color. To this end we can use questions about whether two elements have the same color or not. This
problem is known as the *majority problem*. Saks and Werman [60] showed that the minimum number of questions for adaptive algorithms solving the majority problem is equal to \( n - B(n) \) where \( B(n) \) is the number of 1’s in the binary representation of \( n \). Aigner [61] proved that the minimum number of questions for non-adaptive algorithms solving the majority problem is equal to \( n - q(n) \) where \( q(n) = 2 \) if \( n \) is odd and \( q(n) = 1 \) if \( n \) is even. Adaptive algorithms are algorithms that can decide their work based on each answer of a given question. On the other hand, non-adaptive (oblivious) algorithms depend only on the collective answers of a set of predetermined questions asked.

We consider a modified majority problem: given an \( n \)-tuple of elements colored by one of two colors, we should find an element of the majority color with minimum index if it exists or report that there is no majority color. We studied this problem experimentally and formulated a hypothesis based on the results of experiments. We proved that the minimum number of questions for solving the modified majority problem is equal to \( n - 1 \) both for adaptive and non-adaptive algorithms.

### 6.2.2 Main Notions and Results

Let \( \bar{a} = (a_1, \ldots, a_n) \in \{0, 1\}^n \) be an input of the modified majority problem (\( a_i \) represents the color of \( i \)-th element), and \( S(\bar{a}) = a_1 + \ldots + a_n \). If \( S(\bar{a}) = n/2 \) then the solution of the considered problem \( mmp(\bar{a}) \) is \(-1\). If \( S(\bar{a}) < n/2 \) then \( mmp(\bar{a}) \) is the minimum index \( i \in \{1, \ldots, n\} \) such that \( a_i = 0 \). If \( S(\bar{a}) > n/2 \) then \( mmp(\bar{a}) \) is the minimum index \( i \in \{1, \ldots, n\} \) such that \( a_i = 1 \).

To solve this problem we use decision trees (adaptive algorithms) and tests (non-adaptive algorithms) with attributes (questions) \( g_{i,j} \) where \( i, j \in \{1, \ldots, n\}, i \neq j \) and

\[
g_{i,j}(a_1, \ldots, a_n) = \begin{cases} 0, & a_i \neq a_j, \\ 1, & a_i \neq a_j. \end{cases}
\]
For any $a \in \{0, 1\}$, we define the value $\neg a$ in the following way: $\neg a = 0$ if $a = 1$, and $\neg a = 1$ if $a = 0$. It is clear that $g_{i,j}(a_1, \ldots, a_n) = g_{i,j}(\neg a_1, \ldots, \neg a_n)$ for any $(a_1, \ldots, a_n) \in \{0, 1\}^n$ and $i, j \in \{1, \ldots, n\}, i \neq j$. So as the set of inputs for the modified majority problem we will consider not the set $\{0, 1\}^n$ but the set $In(n) = \{1\} \times \{0, 1\}^{n-1}$.

To study decision trees and tests, we consider decision table $Q_n$ with $n^2 - n$ columns corresponding to the attributes $g_{i,j}, i, j \in \{1, \ldots, n\}, i \neq j$ and $2^{n-1}$ rows corresponding to $n$-tuples from $In(n)$. The row corresponding to a tuple $\bar{a} \in In(n)$ is filled by values of attributes $g_{i,j}$ on $\bar{a}$ and is labeled with the decision $mmp(\bar{a})$ – the solution of the modified majority problem for the input $\bar{a}$.

We study decision trees for $Q_n$, and we denote by $DT(Q_n)$ the set of decision trees for $Q_n$ and by $h(Q_n)$ the minimum depth of a decision tree from the set $DT(Q_n)$. We define $R(Q_n)$ as the minimum cardinality of a test for $Q_n$. It is easy to show that $h(Q_n) \leq R(Q_n)$ (see Corollary 2.24 from [48]): we can construct a decision tree for $Q_n$ which sequentially compute values of attributes from a test for $Q_n$ with minimum cardinality. The depth of this decision tree is equal to $R(Q_n)$.

For $n = 2, \ldots, 10$, we constructed the decision table $Q_n$ and computed the value $h(Q_n)$ which was equal to $n - 1$. The obtained experimental results allowed us to formulate the following hypothesis: $h(Q_n) = R(Q_n) = n - 1$ for any $n \geq 2$. We prove now that this hypothesis is true.

**Theorem 27.** For any $n \geq 2$, the following inequality holds:

$$R(Q_n) \leq n - 1.$$ 

**Proof.** We prove now that the set $\{g_{1,2}, \ldots, g_{1,n}\}$ is a test for the table $Q_n$. Let $\bar{a} = (1, a_2, \ldots, a_n) \in In(n)$. Then the tuple $(g_{1,2}(\bar{a}), \ldots, g_{1,n}(\bar{a}))$ is equal to $(a_2, \ldots, a_n)$. Therefore all rows of $Q_n$ are pairwise different on the columns $g_{1,2}, \ldots, g_{1,n}$, and
\{g_{1,2}, \ldots, g_{1,n}\} is a test for $Q_n$. \hfill \Box

**Theorem 28.** For any $n \geq 2$, the following inequality holds:

$$h(Q_n) \geq n - 1.$$ 

**Proof.** Let $\Gamma \in DT(Q_n)$, and $p(n) = 1$ if $n$ is odd and 0 otherwise. It is easy to see that there exists exactly one tuple $\bar{\gamma} = (1, \gamma_2, \ldots, \gamma_n) \in In(n)$ for which $mmp(\bar{\gamma}) = \frac{n+p(n)}{2}$.

In the tuple $\bar{\gamma}$, the first $\frac{n+p(n)}{2} - 1$ digits are equal to 1 and all other digits are equal to 0. Therefore $Q_n$ contains exactly one row labeled with the decision $\frac{n+p(n)}{2}$, and $\Gamma$ contains a terminal node labeled with the decision $\frac{n+p(n)}{2}$.

We study the path $\pi$ from the root of $\Gamma$ to the terminal node labeled with $\frac{n+p(n)}{2}$. Since $n \geq 2$, the root of $\Gamma$ is not a terminal node. Let $\pi$ contain $m$ nonterminal nodes $v_1, \ldots, v_m$ labeled with attributes $g_{i_1,j_1}, \ldots, g_{i_m,j_m}$, and $m$ edges starting in nodes $v_1, \ldots, v_m$ and labeled with numbers $b_1, \ldots, b_m$, respectively. Since $\Gamma \in DT(Q_n)$, the subtable $Q'_n = Q_n(g_{i_1,j_1}, b_1) \ldots (g_{i_m,j_m}, b_m)$ is not empty and contains only one row corresponding to the tuple $\bar{\gamma}$.

We consider undirected graph $G$ containing nodes $1, \ldots, n$ and edges

$$\{i_1, j_1\}, \ldots, \{i_m, j_m\}.$$ 

Let us assume that the graph $G$ is not connected. Let $C$ be the set of nodes of the connected component of $G$ containing the node 1, and $D = \{1, \ldots, n\} \setminus C$. Without loss of generality, we can assume that $C = \{1, \ldots, k\}$ and $D = \{k+1, \ldots, n\}$ for some $k \in \{1, \ldots, n-1\}$. It is clear that the tuple $\bar{\gamma} = (1, \gamma_2, \ldots, \gamma_n)$ is a solution of the system of equations

$$\{g_{i_1,j_1}(\bar{x}) = b_1, \ldots, g_{i_m,j_m}(\bar{x}) = b_m\}.$$
Since there are no edges connecting sets of nodes $C$ and $D$ in the graph $G$, the tuple $(1, \gamma_2, \ldots, \gamma_k, \neg\gamma_{k+1}, \ldots, \neg\gamma_n)$ is also a solution of the considered system of equations. Therefore $Q'_n$ contains at least two rows which is impossible. Thus the graph $G$ is connected and contains at least $n - 1$ edges. As a result, the length of $\pi$ is at least $n - 1$ and the depth $\Gamma$ is at least $n - 1$.

The next statement follows immediately from Theorems 27 and 28, and from the inequality $h(Q_n) \leq R(Q_n)$.

**Corollary 29.** For any $n \geq 2$, the following equalities hold

$$h(Q_n) = R(Q_n) = n - 1.$$
Chapter 7

Optimization of Decision Tests

This chapter discusses the problem of finding minimum tests for a given decision table. Section 7.1 starts with an introduction to the minimum test problem. We then review some of the important notions related to the problem and present an algorithm to find minimum tests in Section 7.2. We show the results of experiments on datasets from the UCI Machine Learning Repository [20] in Section 7.3.

7.1 Introduction

Decision tests have many applications in various fields. For example, they can model non-adaptive algorithms as discussed in Section 6.2. Finding minimum decision tests allows us to find optimal non-adaptive algorithms for different problems. In addition, tests (superreducts) are key objects in the rough set theory [17–19]. They are also related to knowledge discovery, feature selection and data mining.

It is well known that finding minimum tests for a decision table is NP-hard as the set cover problem can be reduced to it [48]. Let $SC = (U, S)$ be an instance of a set cover problem where $U = \{e_1, \ldots, e_n\}$ is a universe of elements and $S = \{S_1, \ldots, S_m\}$ is a collection of subsets of $U$. We consider $S' \subseteq S$ as a cover of $U$ if each element $e$ from $U$ belongs to at least one subset from $S'$. We create a decision table $T_{SC}$ whose minimum test corresponds to a minimum cover of $SC$. $T_{SC}$ is a decision table with $n + 1$ rows and $m$ columns. The first $n$ rows are constructed as follows. The intersection of row $i$, $i = 1, \ldots, n$, with column $j$ where $j = 1, \ldots, m$ is 1 iff element
\( e_i \) belongs to subset \( S_j \) and 0 otherwise. Row \( n + 1 \) is filled with zeros. We assign a decision 0 to the first \( n \) rows and 1 to row \( n + 1 \). It is clear that a test for this table should separate each row \( i, i = 1, \ldots, n \), from row \( n + 1 \) and must include a column whose value at row \( i \) is 1. Hence, any subset of columns \( S' \) that is a cover for \( SC \) corresponds to a test for \( T_{SC} \) and vice versa.

The problem of finding minimal support sets for a partially defined Boolean function is an important problem in the logical analysis of data (LAD). It is equivalent to the problem of finding minimal decision tests (reducts). Different approaches have been proposed to find minimal support sets utilizing different set cover formulations [62, 63]. In addition, the problem of finding a minimum support set is reduced to the problem of finding a minimum prime implicant of a Boolean function in [64].

Many approximation approaches for optimization of tests have been presented in literature [65, 66]. However, based on results of Feige for the set cover problem [67], it is possible to show that, under some natural assumptions about the class \( NP \), the approximation ratio of the best approximate polynomial algorithm for tests optimization is near to the natural logarithm on the number of pairs of rows (objects) with different decisions in the decision table [68]. Based on results of Dinur and Steurer [69], we can prove the same statement under the assumption that \( P \neq NP \). Therefore, the improvement of exact algorithms for reduct optimization continue to be an important issue.

This chapter presents an algorithm based on dynamic programming that finds a minimum test of a given decision table [16]. This algorithm works by transforming the original decision table \( T \) into a special one \( T^{(1)} \) whose minimum test is also a minimum test for the original table. It then simplifies the new table based on some reduction rules and obtains a subset \( A \) of attributes of \( T^{(1)} \) in addition to a simplified table \( T^{(2)} \). The algorithm concludes its work by finding the remaining attributes of a minimum test through reducing the problem to a problem of decision tree optimization.
In many cases, the simplification algorithm finds a minimum test for $T^{(1)}$ and $T^{(2)}$ is degenerate in that case. We finally present the results of experiments of applying the algorithm on datasets from UCI Machine Learning Repository [20].

7.2 Methodology

This section starts by introducing notions related to our problem and provides an example illustrating these notions. It then describes the proposed algorithm for finding a minimum decision test for a given decision table.

7.2.1 Main Notions

To review, we are given a decision table $T$ with $n$ conditional attributes $f_1, \ldots, f_n$ and one decision attribute $d(T)$. We denote by $U(T)$ the set of rows of $T$, by $A(T)$ the set of conditional attributes of $T$ and by $P(T)$ the set of unordered pairs $\{\rho', \rho''\}$ of rows of $T$ with different decisions.

A conditional attribute $f \in A(T)$ is said to separate a pair of rows with different decisions $\{\rho', \rho''\} \in P(T)$ if those rows have different values of the attribute $f$. We denote by $SEP(T)$ the set of separable subtables of $T$. In the worst case, $|SEP(T)|$ is exponential in the number of attributes of $T$.

A test (superreduct) for $T$ is a subset of columns (conditional attributes) $C \subseteq A(T)$ such that any pair of rows with different decisions of $T$ are separated by at least an attribute of $C$. A reduct for $T$ is a minimal test for $T$, i.e., it is a test for which each proper subset of its attributes is not a test for $T$. We denote by $R(T)$ the minimum cardinality of a reduct for $T$. Reducts for $T$ with cardinality $R(T)$ will be called optimal. The notion of an optimal reduct is equivalent to the notion of an optimal test.

We illustrate the previous notions by the example decision table $T$. $T$ has three rows and three conditional attributes. $U(T) = \{r_1, r_2, r_3\}$, $A(T) = \{f_1, f_2, f_3\}$ and
\[ P(T) = \{\{r_1, r_3\}, \{r_2, r_3\}\}. \] T has five possible tests

\[ \{f_3\}, \{f_1, f_2\}, \{f_1, f_3\}, \{f_2, f_3\}, \{f_1, f_2, f_3\}. \]

It has only two reducts: \(\{f_3\}\) and \(\{f_1, f_2\}\).

\[
T = \begin{array}{cccc}
  & f_1 & f_2 & f_3 & d \\
r_1 & 0 & 1 & 0 & 0 \\
r_2 & 1 & 0 & 0 & 0 \\
r_3 & 1 & 1 & 1 & 1 \\
\end{array}
\]

For \(a = (a_1, \ldots, a_k), b = (b_1, \ldots, b_k) \in \{0, 1\}^k\), we will write \(a \leq b\) if \(a_1 \leq b_1, \ldots, a_k \leq b_k\).

### 7.2.2 Algorithm

We are ready to describe an algorithm for the construction of an optimal reduct.

First, the algorithm transforms the decision table \(T\) into a decision table \(T^{(1)}\) which has \(n\) columns labeled with the conditional attributes \(f_1, \ldots, f_n\), and \(|P(T)| + 1\) rows. The first \(|P(T)|\) rows \(r_1, \ldots, r_{|P(T)|}\) are filled by 0s and 1s, and correspond to unordered pairs \(\{\rho', \rho''\}\) of rows of \(T\) with different decisions. The row of \(T^{(1)}\) corresponding to a pair \(\{\rho', \rho''\}\) contains 1 at the intersection with the column \(f_i, i = 1, \ldots, n\), if and only if \(\rho'\) and \(\rho''\) have different values in the column \(f_i\), i.e., \(f_i\) separates this pair of rows. The last row \(r_{|P(T)|+1}\) in \(T^{(1)}\) is filled by 0s. The last row is labeled with the decision 1. All other rows are labeled with the decision 0.

Each reduct of the table \(T^{(1)}\) must contain an attribute that separates each row \(r_i, 1 \leq i \leq |P(T)|\) from \(r_{|P(T)|+1}\). Since the value of all attributes on the row \(r_{|P(T)|+1}\) is zero, then any reduct of \(T^{(1)}\) contains one or more attributes with value 1 for all other rows. The following statement is almost obvious.

**Proposition 30.** Decision tables \(T\) and \(T^{(1)}\) have the same set of reducts.
Algorithm $A_6$ (Simplification Algorithm)

Input: A decision table $T^{(1)}$ with $m = |P(T)| + 1$ rows $r_1, \ldots, r_m$ and $n$ conditional attributes $f_1, \ldots, f_n$.

Output: A subset $A$ of conditional attributes and a decision table $T^{(2)}$.

1. Set $A = \emptyset$.

2. For each pair of columns $f_i$ and $f_j$ of $T^{(1)}$ such that $i \neq j$ and $f_i \leq f_j$, remove the column $f_i$.

3. For each row $r_i$ of $T^{(1)}$, $1 \leq i \leq |P(T)|$, that is separated from the last row by an unique attribute $f_j$, add the attribute $f_j$ to the set $A$ and remove all rows that are separated by this attribute from the row $r_{|P(T)|+1}$.

4. Set $T^{(2)}$ the decision table obtained after step 3.

5. Return $A$ and $T^{(2)}$.

The next step of the algorithm (see Algorithm $A_6$) is to apply two reduction rules in the following order.

1. Reduction rule $R_1$: For each pair of columns $f_i$ and $f_j$ of $T^{(1)}$ such that $i \neq j$ and $f_i \leq f_j$, remove column $f_i$.

2. Reduction rule $R_2$: For each row $r_i$ of $T^{(1)}$, $1 \leq i \leq |P(T)|$ that is separated from the last row by a unique attribute $f'$, add this attribute $f'$ to the partial reduct $A$ and remove all of rows that are separated by this attribute from row $r_{|P(T)|+1}$.

Let $T'$ be the intermediary table obtained after applying $R_1$ on $T^{(1)}$. $R_1$ states that if an attribute $f_j$ separates all pair of rows separated by $f_i$ then we can replace $f_i$ by $f_j$ in any reduct containing $f_i$ and we obtain a reduct with at most the same
number of attributes. As a result, each optimal reduct of $T'$ is an optimal reduct for $T^{(1)}$. $R_2$ describes the fact that if a pair of rows of $T'$ are separated by only one attribute then this attribute must belong to any reduct of $T'$.

It is clear that the space complexity of the simplification algorithm is $O(|P(T)| \times n)$ where $|P(T)|$ is the number of unordered pairs of rows of $T$ with different decisions and $|A(T)| = n$ is the number of conditional attributes of $T$. The first rule of the simplification algorithm has time complexity of $O(|P(T)| \times n^2)$ while time complexity of the second rule is $O(|P(T)| \times n)$. Hence, the time complexity of the simplification algorithm is $O(|P(T)| \times n^2)$.

We denote by $T^{(2)}$ the table obtained after the application of $R_1$ and $R_2$. It is clear that this table contains the row $r_{|P(T)|+1}$. One can prove the following statement.

**Proposition 31.** The union of each optimal reduct for $T^{(2)}$ with the set $A$ obtained after the application of $R_1$ and $R_2$ is an optimal reduct for the table $T$.

The last step of the algorithm is the construction, for the decision table $T^{(2)}$, a decision tree $\Gamma$ with minimum depth. It is not difficult to prove the following statement.

**Proposition 32.** Let $\Gamma$ be a decision tree with minimum depth for $T^{(2)}$. Then, the set $B$ of attributes attached to the path in $\Gamma$ from the root to a terminal node which accepts the row $r_{|P(T)|+1}$ is an optimal reduct for the table $T^{(2)}$.

Therefore, the set $A \cup B$ is an optimal reduct for $T$.

To construct $\Gamma$ we use a dynamic programming algorithm implemented in the system DAGGER [12] which finds a subset of the set of all decision trees with minimum depth for a given decision table. As subproblems of the initial problem (decision table) this algorithm uses separable subtables of the input table. The run time of this algorithm is polynomial in the number of separable subtables of the input table ($|SEP(T^{(2)})|$). In the worst case, this algorithm has exponential time complexity.
relative to the size of decision table. However, it is applicable, usually, to medium size decision tables.

7.3 Experiments

| Decision table   | Size of table $T$ | Size of table $T^{(2)}$ | $|P(T)|$ | $|A|$ | $R(T)$ | Time$^1$ | Time$^2$ |
|------------------|------------------|--------------------------|--------|------|--------|----------|----------|
|                  | #rows | #atts |                  | #rows | #atts |        |          |          |
| adult-stretch    | 16    | 4     | 48               | 0     | 2     | 2      | 2        | 0.001    |
| balance-scale    | 625   | 4     | 111168           | 0     | 0     | 4      | 4        | 0.036    |
| breast-cancer    | 266   | 9     | 14440            | 0     | 1     | 8      | 8        | 0.007    |
| cars             | 1728  | 6     | 682721           | 0     | 0     | 6      | 6        | 0.312    |
| hayes-roth-data  | 69    | 4     | 1548             | 0     | 0     | 4      | 4        | 0.001    |
| house-votes-84   | 279   | 16    | 17204            | 4     | 6     | 10     | 11       | 0.012    |
| kr-vs-kp         | 3196  | 36    | 2548563          | 18    | 9     | 27     | 29       | 6.168    |
| lenses           | 24    | 4     | 155              | 0     | 0     | 4      | 4        | 0.006    |
| lymphography     | 148   | 18    | 5801             | 5801  | 18    | 0      | 6        | 0.003    |
| monks-1-test     | 432   | 6     | 46656            | 0     | 3     | 3      | 3        | 0.021    |
| monks-1-train    | 124   | 6     | 3844             | 0     | 3     | 3      | 3        | 0.002    |
| monks-2-test     | 432   | 6     | 41180            | 0     | 0     | 6      | 6        | 0.016    |
| monks-2-train    | 169   | 6     | 6720             | 0     | 0     | 6      | 6        | 0.002    |
| monks-3-test     | 432   | 6     | 46512            | 0     | 3     | 3      | 3        | 0.011    |
| monks-3-train    | 122   | 6     | 3720             | 0     | 2     | 4      | 4        | 0.001    |
| mushroom         | 8124  | 22    | 16478528         | 16478528 | 19 | 0     | 4      | 75.026   |
|                  |       |       |                  |        |       |        |          | 329.542  |
| nursery          | 12960 | 8     | 57319460         | 0     | 0     | 8      | 8        | 78.86    |
| shuttle-landing  | 15    | 6     | 54               | 0     | 1     | 5      | 5        | 0        |
| soybean-small    | 47    | 35    | 810              | 810   | 14    | 0      | 2        | 0.002    |
| spect-test       | 169   | 22    | 1288             | 11    | 14    | 8      | 11       | 0.185    |
| Teeth            | 23    | 8     | 253              | 0     | 2     | 6      | 6        | 0        |
| tic-tac-toe      | 958   | 9     | 207832           | 207832 | 9 | 0     | 8      | 0.08     |
| zoo-data         | 59    | 16    | 1405             | 105   | 14    | 2      | 5        | 0.002    |

Table 7.1: Characteristics of decision tables and results of experiments.
Size of final decision table $T^{(2)}$ – the number of conditional attributes (#atts) and the number of rows other than the last row (#rows);
Time$^1$ is the time in seconds taken for creating $T^{(1)}$ from $T$ and then transforming it into $T^{(2)}$;
Time$^2$ is the time taken in seconds for finding optimal reducts for $T^{(2)}$ with DAGGER.

In this section, we present experimental results for 23 datasets (decision tables) from UCI ML Repository [20]. Some decision tables contain conditional attributes that take unique value for each row. Such attributes were removed. In some tables there are equal rows with, possibly, different decisions. In this case, each group of
identical rows is replaced with a single row from the group with the most common
decision for this group. In some tables there were missing values. Each such value
was replaced with the most common value of the corresponding attribute.

Table 1 contains information about decision tables and results of experiments:

- name of initial decision table $T$;
- size of decision table $T$ – the number of conditional attributes $\#\text{atts}$ and rows $\#\text{rows}$;
- number $|P(T)|$ such that the number of rows in $T^{(1)}$ is equal to $|P(T)| + 1$ (the
  number of attributes in $T^{(1)}$ is the same as in $T$);
- size of final decision table $T^{(2)}$ – the number of conditional attributes $\#\text{atts}$ and
  rows other than the last row $\#\text{rows}$;
- cardinality $|A|$ of the set of attributes $A$ constructed during the simplification
  of $T^{(1)}$;
- minimum cardinality $R(T)$ of a reduct for $T$;
- $\text{Time}^1$ is the time in seconds taken for creating $T^{(1)}$ from $T$ and then transform-
ing it into $T^{(2)}$;
- $\text{Time}^2$ is the time taken in seconds for finding optimal reducts for $T^{(2)}$ with
  DAGGER.

A Mac Pro desktop with 16 GB of RAM memory and dual Intel(R) Xeon(R)
processors of 2.67 GHz is used for the experiments. Both phases of the algorithm are
run sequentially. The time of each phase is measured on average of ten executions of
this stage for each data set.

For 15 out of the 23 considered data sets, the reduction rules in the simplification
part of the algorithm managed to find a minimum reduct. It also reduced the number
of attributes of other data sets dramatically such as: kr-vs-kp and soybean-small. Data sets solved by the simplification part of the algorithm has zero as the number of rows and/or columns in $T^{(2)}$. Some data sets have zero rows and more than zero columns as our implementation of the algorithm finishes its work once no more rows need to be separated from the last row.

The simplification phase of the algorithm achieved fast runtime for most of the data sets with exception of mushroom and nursery due to the huge size of the table $T^{(1)}$. A simple brute force algorithm may find optimal reducts by considering each possible subset and testing whether it form a reduct or not. Such brute force algorithm would have complexity of $\Omega(2^n \times P(T))$ on table $T$ with $n$ conditional attributes. It will be very difficult to apply this algorithm on data sets such as: kr-vs-kp, mushroom, nursery and soybean-small.

Note that the dynamic programming algorithm is used if $|A| < R(T)$. If $|A| = R(T)$ then an optimal reduct $A$ for $T$ is constructed during the simplification of the table $T^{(1)}$. In such cases, the value of the cell Time$^2$ for this data set is empty.

The dynamic programming algorithm processed $T^{(2)}$ in a considerably fast time for all data sets with exception of lymphography and mushroom due to the large number of separable subtables of each.
Chapter 8

Conclusion

This thesis investigated various questions regarding the complexity of several important computational problems. In most cases, we seek optimal algorithms for the studied problems. Such algorithms are very important for the following reasons. First, they allow us to precisely identify the complexity of the studied problem and to fill the gaps between lower and upper bounds. More importantly, they serve as research tools that may provide useful insights for improving current heuristics for the studied problems or designing new ones especially if the problem is computationally hard.

We adopt a common approach while tackling the considered problems. This approach starts with defining a model of computation as a discrete structure that describes the class of algorithms studied for the given problem. This step is critical as failure to capture members of the class of studied algorithms invalidates the obtained results. Then, we identify the complexity measure under investigation and express it in the form of a cost function. Finally, the last step is to find optimal structures (algorithms) with respect to the considered cost function and with the use of an optimization technique.

This thesis has designed and used various optimization algorithms based on extensions of dynamic programming. This technique is a powerful tool that allows us to represent the search space compactly, to count its size without enumerating it and to find the set of whole optimal objects if the considered cost function is strictly increasing. This calls for considering a multi-stage optimization process. It is particularly useful in the situation where one has various optimization criteria with clear order of
those criteria based on their priorities.

In addition, extensions of dynamic programming provides naturally a way of bi-criteria optimization. This allows us to study relationships and trade-offs between pairs of complexity measures. Identifying such relationships sets up better understanding of what limitations that we cannot overcome.

Although this optimization technique is very powerful, it has exponential time complexity in many cases. This shortcoming enforces us to avoid using the technique in large instances of the considered problems. However, we notice that this limitation should not restrain us from using this tool. We may use different tools of mathematical analysis to generalize insights and results obtained by applying this technique on smaller instances for larger ones.

The first part of the thesis considers algorithms for multi-frontal solvers of the \textit{hp}-adaptive finite element method (FEM). The method is a special type of the traditional finite element method which is used ubiquitously as an analysis tool in different engineering fields.

Chapter 2 mathematically defines the class of 2D rectangular meshes studied and proposes a new model of computation (element partition tree) that represents a class of algorithms used by multi-frontal solvers. We note that the class represented is a smaller class of all possible algorithms as we restrict ordering of elements to those obtained by straight line partitions of the input mesh. We decided to restrict the class of algorithms considered as it is well known that finding optimal algorithms for this problem is NP-hard with respect to some cost functions such as: fill-in [35] and number of flops [42]. This restriction allows us to design and propose multi-stage and bi-criteria optimization techniques for the considered criteria that has polynomial time complexity. On the other hand, a drawback of this restriction is that we can only say about the optimal algorithms found that they are locally optimal (quasi-optimal). Finally, the chapter concludes with an abstract definition of cost functions
and concrete definitions of the studied cost functions.

Different algorithms based on extensions of dynamic programming are highlighted in Chapter 3. Those algorithms allow us to represent the set of possible element partition trees for a given problem, count its cardinality and apply multi-stage optimization of element partition trees with respect to a sequence of cost functions. Complexity of those algorithms are mathematically analyzed and a proof of correctness is provided. The chapter finishes with a sequence of experiments involving finding optimal element partition trees for various benchmark 2D meshes with point and edge singularities. Such results helped in creating efficient heuristics [7,8] for 2D meshes, and were generalized to 3D meshes, that outperform state of the art heuristics such as: nested dissection and minimum degree algorithm. Although we find optimal algorithms for some of the benchmark problems, future research may consider other unconsidered benchmark problems and cost functions.

Chapter 4 concludes the work on multi-frontal solvers for 2D $hp$-adaptive FEM by presenting a bi-criteria optimization algorithm. Different mathematical tools and algorithms are presented and used to create this algorithm. A mathematical analysis of the algorithm complexity in addition to a proof of correctness is shown. We applied this technique to find the set of Pareto optimal points for a given mesh and two cost functions and use it to depict the relationship between this pair of cost functions. We described a way of utilizing these results to improve iterative solvers in [9]. The proposed algorithms can construct a Pareto optimal element partition tree corresponding to a given Pareto optimal point. As future work, further experiments for finding Pareto optimal algorithms (element partition trees) for other benchmark meshes and cost functions may provide insights into designing balanced heuristics. In addition, further relationships between pairs of cost functions can be studied for various classes of meshes.

The second part of the thesis starts by presenting decision tables as a way of
problem representation in Chapter 5. It presents a general problem and how it can be represented in the form of a decision table. The chapter continues by defining decision trees and decision tests for decision tables. It shows how decision trees for a decision table can model adaptive algorithms for the represented problem. In addition, it demonstrates the use of decision tests as models of non-adaptive algorithms.

Chapter 6 studies optimal algorithms for the problem of sorting of eight elements and a variant of the majority problem of an arbitrary size. For sorting, we find that the smallest average depth of a decision tree for sorting of eight distinct elements is $620160/8!$. We prove that each decision tree for the considered problem that has this average depth has also minimum depth. Those questions were discussed by Knuth in [11]. For the variant of majority problem, we proved that any correct algorithm for solving the considered variant of the majority problem with $n$ elements can not solve it in less than $n - 1$ questions in both adaptive and non-adaptive cases and presented one such algorithm.

The last model of computation considered in this thesis is decision test. It can be used to model non-adaptive algorithms in a natural way. Chapter 7 introduces a dynamic programming based algorithm for finding a minimum decision test of a given decision table. This optimization problem is NP-hard so the considered algorithm runs in exponential time in the worst-case. We apply the algorithm on 23 datasets from the UCI Machine Learning Repository [20]. We discussed the results and runtime of our algorithm and we noted that the algorithm finds a minimum test for 15 out of the 23 considered datasets using only the simplification part of the algorithm which runs in polynomial time.

Decision tests have other applications in feature selection, knowledge representation and data mining. As a result, it is important to consider heuristics that can find small tests for very large data efficiently. We propose the design of efficient approximation algorithms that can run for large data as future work. Moreover, having such
efficient algorithm encourages several applications of reducts in different fields such as: data mining, feature selection and knowledge representation.
REFERENCES


APPENDICES

A Thesis Publications


B Other Publications


C  Publications Submitted and Under Preparation

